

Index

Sr.No	Topic	Date	Remark
1	Practical 1 : Write the following programs for Blockchain in Python : (I). A simple client class that generates the private and public keys by using the built in Python RSA algorithm and test it (II). A transaction class to send and receive money and test it .		
2	Practical 2 : Write the following programs for Blockchain in Python : (I).Create multiple transactions and display them . (II). Create a blockchain, a genesis block and execute it .		
3	Practical 3 : Write the following programs for Blockchain in Python : (I).Create a mining function and test it . (II).Add blocks to the miner and dump the blockchain .		
4	Practical 4 : Implement and demonstrate the use of the following in Solidity : (I).Varaible (II).Operators (III).Loops (IV).Decision Making (V).Strings		
5	Practical 5 : Implement and demonstrate the use of the following in Solidity : (I).Arrays (II).Enums (III).Structs (IV).Mappings (V).Coversations (VI).Ether Units (VII).Special Variables		

Index

Sr.No	Topic	Date	Remark
6	Practical 6: Implement and demonstrate the use of the following in Solidity :		
	(I).Functions		
	(II).View Functions		
	(III).Pure Functions		
	(IV).Fallback Functions		
	(V).Function Overloading		
	(VI).Mathematical Functions		
	(VII).Cryptographic Functions		
7	Practical 7 : Implement and demonstrate the use of the following in Solidity :		
	(I).Contracts		
	(II).Inheritance		
	(III).Constructors		
	(IV).Abstract Class		
	(V).Interfaces		
8	Practical 8 : Implement and demonstrate the use of the following in Solidity :		
	(I).Libraries		
	(II).Assembly		
	(III).Events		
	(IV).Error Handling		

Practical 1

Write the following programs for Blockchain in Python :

(I) A simple client class that generates the private and public keys by using the built in Python RSA algorithm and test it

(II) A transaction class to send and receive money and test it .

→

```
# import libraries

import hashlib
import random
import string
import json
import binascii
import numpy as np
import pandas as pd
import pylab as pl
import logging
import datetime
import collections
pip install pycryptodome

# following imports are required by PKI

import Crypto
import Crypto.Random
from Crypto.Hash import SHA
from Crypto.PublicKey import RSA
from Crypto.Signature import PKCS1_v1_5

import binascii

class Client:

    def __init__(self):
```

```

random = Crypto.Random.new().read
self._private_key = RSA.generate(1024, random)
self._public_key = self._private_key.publickey()
self._signer = PKCS1_v1_5.new(self._private_key)

@property
def identity(self):
    return binascii.hexlify(self._public_key.exportKey(format='DER')).decode('ascii')

class Transaction:
    def __init__(self, sender, recipient, value):
        self.sender = sender
        self.recipient = recipient
        self.value = value
        self.time = datetime.datetime.now()

    def to_dict(self):
        if self.sender == "Genesis":
            identity = "Genesis"
        else:
            identity = self.sender.identity
        return collections.OrderedDict({
            'sender': identity,
            'recipient': self.recipient,
            'value': self.value,
            'time': self.time})

    def sign_transaction(self):
        private_key = self.sender._private_key
        signer = PKCS1_v1_5.new(private_key)
        h = SHA.new(str(self.to_dict()).encode('utf8'))

```

```
return binascii.hexlify(signer.sign(h)).decode('ascii')

Dinesh = Client()

Ramesh = Client()

t = Transaction(Dinesh,Ramesh.identity,5.0)

signature = t.sign_transaction()

print (signature)
```

Output:

The screenshot shows a Jupyter Notebook interface with the following details:

- Title:** BLOCK CHAIN pract 1(A).ipynb
- Toolbar:** File, Edit, View, Insert, Runtime, Tools, Help, Saving...
- Code Cell 2:** Displays Python code imports:

```
[2] # import libraries
import hashlib
import random
import string
import json
import binascii
import numpy as np
import pandas as pd
import pylab as pl
import logging
import datetime
import collections
```
- Code Cell 3:** Displays a pip install command and its output:

```
[3] pip install pycryptodome

Collecting pycryptodome
  Downloading https://files.pythonhosted.org/packages/ad/16/9627ab0493894a11c68e46000dbcc82f578c8ff06bc2980dc016aea9bd3/pycryptodome-3.10.1-cp35-abi3-manylinux2
    1.9MB 4.8MB/s
Installing collected packages: pycryptodome
Successfully installed pycryptodome-3.10.1
```

BLOCK CHAIN pract 1(A).ipynb

```
[5] import Crypto
import Crypto.Random
from Crypto.Hash import SHA
from Crypto.PublicKey import RSA
from Crypto.Signature import PKCS1_v1_5

import binascii

import binascii
class Client:
    def __init__(self):
        random = Crypto.Random.new().read
        self._private_key = RSA.generate(1024, random)
        self._public_key = self._private_key.publickey()
        self._signer = PKCS1_v1_5.new(self._private_key)

    @property
    def identity(self):
        return binascii.hexlify(self._public_key.exportKey(format='DER')).decode('ascii')

class Transaction:
    def __init__(self, sender, recipient, value):
        self.sender = sender
        self.recipient = recipient
        self.value = value
        self.time = datetime.datetime.now()
```

Waiting for colab.research.google.com...

BLOCK CHAIN pract 1(A).ipynb

```
self.value = value
self.time = datetime.datetime.now()
def to_dict(self):
    if self.sender == "Genesis":
        identity = "Genesis"
    else:
        identity = self.sender.identity
    return collections.OrderedDict([
        ('sender': identity,
         'recipient': self.recipient,
         'value': self.value,
         'time': self.time)])
def sign_transaction(self):
    private_key = self.sender._private_key
    signer = PKCS1_v1_5.new(private_key)
    h = SHA.new(str(self.to_dict()).encode('utf8'))
    return binascii.hexlify(signer.sign(h)).decode('ascii')

Dinesh = Client()
Ramesh = Client()

t = Transaction(Dinesh,Ramesh.identity,5.0)

signature = t.sign_transaction()
print (signature)
```

656f0e37653d1bc3c538de01595a9c7e4a603e03740d1ed450f1ea32f568663cebff7ec9757cdd35d4f8b1a3524341528a48674981265b68a477c06d57271d6250f45d6e3946a4f8166fff50251956fa

Waiting for colab.research.google.com...

Practical 2

Write the following programs for Blockchain in Python :

- (I).Create multiple transactions and display them .
- (II). Create a blockchain, a genesis block and execute it .

→

```
def display_transaction(transaction):  
    #for transaction in transactions:  
    dict = transaction.to_dict()  
    print ("sender: " + dict['sender'])  
    print (' ----')  
    print ("recipient: " + dict['recipient'])  
    print (' ----')  
    print ("value: " + str(dict['value']))  
    print ('-----')  
    print ("time: " + str(dict['time']))  
    print ('-----')
```

```
transactions = []
```

```
Dinesh = Client()
```

```
Ramesh = Client()
```

```
Seema = Client()
```

```
Vijay = Client()
```

```
t1 = Transaction(
```

```
    Dinesh,
```

```
    Ramesh.identity,
```

```
    15.0
```

)

```
t1.sign_transaction()  
transactions.append(t1)
```

t2 = Transaction(

 Dinesh,

 Seema.identity,

 6.0

)

```
t2.sign_transaction()  
transactions.append(t2)
```

t3 = Transaction(

 Ramesh,

 Vijay.identity,

 2.0

)

```
t3.sign_transaction()  
transactions.append(t3)
```

t4 = Transaction(

 Seema,

 Ramesh.identity,

 4.0

)

```
t4.sign_transaction()  
transactions.append(t4)
```

t5 = Transaction(

 Vijay,

 Seema.identity,

```
    7.0
)
t5.sign_transaction()
transactions.append(t5)

t6 = Transaction(
    Ramesh,
    Seema.identity,
    3.0
)
t6.sign_transaction()
transactions.append(t6)

t7 = Transaction(
    Seema,
    Dinesh.identity,
    8.0
)
t7.sign_transaction()
transactions.append(t7)

t8 = Transaction(
    Seema,
    Ramesh.identity,
    1.0
)
t8.sign_transaction()
transactions.append(t8)

t9 = Transaction(
    Vijay,
    Dinesh.identity,
    5.0
```

```
)  
t9.sign_transaction()  
transactions.append(t9)  
t10 = Transaction(  
    Vijay,  
    Ramesh.identity,  
    3.0  
)  
t10.sign_transaction()  
transactions.append(t10)
```

for transaction in transactions:

```
    display_transaction (transaction)  
    print ('-----')
```

class Block:

```
    def __init__(self):  
        self.verified_transactions = []  
        self.previous_block_hash = ""  
        self.Nonce = ""
```

last_block_hash = ""

Dinesh = Client()

```
t0 = Transaction (  
    "Genesis",  
    Dinesh.identity,  
    500.0
```

```
)  
  
block0 = Block()  
  
block0.previous_block_hash = None  
Nonce = None  
  
block0.verified_transactions.append (t0)  
  
digest = hash (block0)  
last_block_hash = digest  
  
TPCoins = []  
  
def dump_blockchain (self):  
    print ("Number of blocks in the chain: " + str(len (self)))  
    for x in range (len(TPCoins)):  
        block_temp = TPCoins[x]  
        print ("block # " + str(x))  
        for transaction in block_temp.verified_transactions:  
            display_transaction (transaction)  
            print ('-----')  
        print ('=====')  
  
TPCoins.append (block0)  
dump_blockchain(TPCoins)
```

Output:

```
[10] def display_transaction(transaction):
    #for transaction in transactions:
    dict = transaction.to_dict()
    print ("sender: " + dict['sender'])
    print ('-----')
    print ("recipient: " + dict['recipient'])
    print ('-----')
    print ("value: " + str(dict['value']))
    print ('-----')
    print ("time: " + str(dict['time']))
    print ('-----')

[11] transactions = []

[12] Dinesh = Client()
Ramesh = Client()
Seema = Client()
Vijay = Client()

[13] t1 = Transaction(
    Dinesh,
    Ramesh.identity,
    15.0
)

[15] t1.sign_transaction()
transactions.append(t1)

[16] t2 = Transaction(
    Dinesh,
    Seema.identity,
    6.0
)
t2.sign_transaction()
transactions.append(t2)
t3 = Transaction(
    Ramesh,
    Vijay.identity,
    2.0
)
t3.sign_transaction()
transactions.append(t3)
t4 = Transaction(
    Seema,
    Ramesh.identity,
    4.0
)
t4.sign_transaction()
transactions.append(t4)
t5 = Transaction(
    Vijay,
    Seema.identity,
    7.0
)
t5.sign_transaction()
transactions.append(t5)
t6 = Transaction(
    Ramesh,
    Seema.identity,
    3.0
)
t6.sign_transaction()
transactions.append(t6)
t7 = Transaction(
    Seema,
    Dinesh.identity,
    8.0
)
t7.sign_transaction()
transactions.append(t7)
t8 = Transaction(
    Seema,
    Ramesh.identity,
    1.0
)
t8.sign_transaction()
transactions.append(t8)
t9 = Transaction(
    Vijay,
    Dinesh.identity,
    5.0
)
```

```

[16] t8.sign_transaction()
transactions.append(t8)
t9 = Transaction(
    Vijay,
    Dinesh.identity,
    5.0
)
t9.sign_transaction()
transactions.append(t9)
t10 = Transaction(
    Vijay,
    Ramesh.identity,
    3.0
)
t10.sign_transaction()
transactions.append(t10)

[17] for transaction in transactions:
    display_transaction(transaction)
    print('-----')

sender: 30819f300d06092a864886f70d010101050003818d0030818
-----
recipient: 30819f300d06092a864886f70d010101050003818d0030818902818100b5c36acef717c85079d5
-----
value: 15.0
-----
time: 2021-05-05 07:37:46.751762
-----
sender: 30819f300d06092a864886f70d010101050003818d003081890281810096e3e436b160bf2feecba6d
-----
recipient: 30819f300d06092a864886f70d010101050003818d0030818902818100b5c36acef717c85079d5
-----
value: 15.0
-----
time: 2021-05-05 07:37:46.751762
-----
sender: 30819f300d06092a864886f70d010101050003818d003081890281810096e3e436b160bf2feecba6d
-----
recipient: 30819f300d06092a864886f70d010101050003818d0030818902818100a3b073014e3d9a1e3353
-----
value: 6.0
-----
time: 2021-05-05 07:38:17.042489
-----
sender: 30819f300d06092a864886f70d010101050003818d0030818902818100b5c36acef717c85079d5eb8
-----
recipient: 30819f300d06092a864886f70d010101050003818d0030818902818100967bf6965c7e25f7c711
-----
value: 2.0
-----
time: 2021-05-05 07:38:17.044049
-----
sender: 30819f300d06092a864886f70d010101050003818d0030818902818100a3b073014e3d9a1e335365f
-----
recipient: 30819f300d06092a864886f70d010101050003818d0030818902818100b5c36acef717c85079d5
-----
value: 4.0
-----
time: 2021-05-05 07:38:17.045503
-----
sender: 30819f300d06092a864886f70d010101050003818d0030818902818100967bf6965c7e25f7c711a30
-----
recipient: 30819f300d06092a864886f70d010101050003818d0030818902818100a3b073014e3d9a1e3353
-----
value: 7.0
-----
time: 2021-05-05 07:38:17.046076

[18] class Block:
    def __init__(self):
        self.verified_transactions = []
        self.previous_block_hash = ""
        self.Nonce = ""

[19] last_block_hash = ""

```



+ Code + Text

✓ RAM Disk | Editing | ^

```
[20] Dinesh = Client()

[21] t0 = Transaction (
    "Genesis",
    Dinesh.identity,
    500.0
)

[22] block0 = Block()

[23] block0.previous_block_hash = None
    Nonce = None

[24] block0.verified_transactions.append (t0)

[25] digest = hash (block0)
    last_block_hash = digest

[26] def dump_blockchain (self):
    print ("Number of blocks in the chain: " + str(len (self)))
    for x in range (len(TPCoins)):
        block_temp = TPCoins[x]
        print ("block # " + str(x))
        for transaction in block_temp.verified_transactions:
            display_transaction (transaction)
            print ('-----')
    print ('=====')
```

[27] TPCoins = []

[28] def dump_blockchain (self):
 print ("Number of blocks in the chain: " + str(len (self)))
 for x in range (len(TPCoins)):
 block_temp = TPCoins[x]
 print ("block # " + str(x))
 for transaction in block_temp.verified_transactions:
 display_transaction (transaction)
 print ('-----')
 print ('=====')

[29] TPCoins.append (block0)

[30] dump_blockchain(TPCoins)

Number of blocks in the chain: 1
block # 0
sender: Genesis

recipient: 30819f300d06092a864886f70d010101050003818d0030818902818100df7d100622cc76eab161

value: 500.0

time: 2021-05-05 07:41:01.767902

=====

Practical 3

Write the following programs for Blockchain in Python :

- (I).Create a mining function and test it .
- (II).Add blocks to the miner and dump the blockchain .

→

```
def sha256(message):  
    return hashlib.sha256(message.encode('ascii')).hexdigest()  
  
def mine(message, difficulty=1):  
    assert difficulty >= 1  
    prefix = '1' * difficulty  
    for i in range(1000):  
        digest = sha256(str(hash(message)) + str(i))  
        if digest.startswith(prefix):  
            print ("after " + str(i) + " iterations found nonce: " + digest)  
            return digest  
  
last_transaction_index = 0  
  
block = Block()  
for i in range(3):  
    temp_transaction = transactions[last_transaction_index]  
    # validate transaction  
    # if valid  
    block.verified_transactions.append (temp_transaction)  
    last_transaction_index += 1 mine ("test message", 2)
```

```
block.previous_block_hash = last_block_hash  
block.Nonce = mine(block, 2)  
digest = hash(block)  
TPCoins.append(block)  
last_block_hash = digest  
  
# Miner 2 adds a block  
block = Block()  
  
for i in range(3):  
    temp_transaction = transactions[last_transaction_index]  
    # validate transaction  
    # if valid  
    block.verified_transactions.append(temp_transaction)  
    last_transaction_index += 1  
    block.previous_block_hash = last_block_hash  
    block.Nonce = mine(block, 2)  
    digest = hash(block)  
    TPCoins.append(block)  
    last_block_hash = digest  
# Miner 3 adds a block  
block = Block()  
  
for i in range(3):  
    temp_transaction = transactions[last_transaction_index]  
    #display_transaction(temp_transaction)  
    # validate transaction  
    # if valid
```

```
block.verified_transactions.append (temp_transaction)
last_transaction_index += 1
```

```
block.previous_block_hash = last_block_hash
block.Nonce = mine (block, 2)
digest = hash (block)
```

```
TPCoins.append (block)
last_block_hash = digest
dump_blockchain(TPCoins)
```

Full Output:

```

[ ] # import libraries
import hashlib
import random
import string
import json
import binascii
import numpy as np
import pandas as pd
import pylab as pl
import logging
import datetime
import collections

[ ] pip install pycryptodome
Collecting pycryptodome
  Downloading https://files.pythonhosted.org/packages/ad/16/9627ab0493894a11c68e46000dbcc_
[ ] Installing collected packages: pycryptodome
Successfully installed pycryptodome-3.10.1

[ ] # following imports are required by PKI
import Crypto
import Crypto.Random
from Crypto.Hash import SHA
from Crypto.PublicKey import RSA
from Crypto.Signature import PKCS1_v1_5

[ ] import binascii
class Client:
    def __init__(self):
        random = Crypto.Random.new().read
        self._private_key = RSA.generate(1024, random)
        self._public_key = self._private_key.publickey()
        self._signer = PKCS1_v1_5.new(self._private_key)

    @property
    def identity(self):
        return binascii.hexlify(self._public_key.exportKey(format='DER')).decode('ascii')

class Transaction:
    def __init__(self, sender, recipient, value):
        self.sender = sender
        self.recipient = recipient
        self.value = value
        self.time = datetime.datetime.now()
    def to_dict(self):
        if self.sender == "Genesis":
            identity = "Genesis"
        else:
            identity = self.sender.identity
        return collections.OrderedDict({
            'sender': identity,
            'recipient': self.recipient,
            'value': self.value,
            'time' : self.time})
    def sign_transaction(self):
        private_key = self.sender._private_key
        signer = PKCS1_v1_5.new(private_key)
        h = SHA.new(str(self.to_dict()).encode('utf8'))
        return binascii.hexlify(signer.sign(h)).decode('ascii')

[ ]

[ ] Dinesh = Client()
Ramesh = Client()

[ ] t = Transaction(Dinesh,Ramesh.identity,5.0)

[ ] signature = t.sign_transaction()
print (signature)
251f28f6f523733739979611b404c755210b5b6a70f28d5eb3e3049f7dceea873e472f0df41a55152c71bb6f5

[ ] def display_transaction(transaction):
    #for transaction in transactions:
    dict = transaction.to_dict()
    print ('sender: ' + dict['sender'])
    print ('-----')

```

```

[ ] def display_transaction(transaction):
    #for transaction in transactions:
    dict = transaction.to_dict()
    print ("sender: " + dict['sender'])
    print ('-----')
    print ("recipient: " + dict['recipient'])
    print ('-----')
    print ("value: " + str(dict['value']))
    print ('-----')
    print ("time: " + str(dict['time']))
    print ('-----')

[ ] transactions = []

[ ] Dinesh = Client()
Ramesh = Client()
Seema = Client()
Vijay = Client()

[ ] t1 = Transaction(
    Dinesh,
    Ramesh.identity,
    15.0
)

[ ] t1.sign_transaction()
transactions.append(t1)

[ ] t2 = Transaction(
    Dinesh,
    Seema.identity,
    6.0
)
t2.sign_transaction()
transactions.append(t2)
t3 = Transaction(
    Ramesh,
    Vijay.identity,
    2.0
)
t3.sign_transaction()
transactions.append(t3)
t4 = Transaction(
    Seema,
    Ramesh.identity,
    4.0
)
t4.sign_transaction()
transactions.append(t4)
t5 = Transaction(
    Vijay,
    Seema.identity,
    7.0
)
t5.sign_transaction()
transactions.append(t5)
t6 = Transaction(
    Ramesh,
    Seema.identity,
    3.0
)
t6.sign_transaction()
transactions.append(t6)
t7 = Transaction(
    Seema,
    Dinesh.identity,
    8.0
)
t7.sign_transaction()
transactions.append(t7)
t8 = Transaction(
    Seema,
    Ramesh.identity,
    1.0
)
t8.sign_transaction()
transactions.append(t8)
t9 = Transaction(
    Vijay,
    Dinesh.identity,
    5.0
)
t9.sign_transaction()
transactions.append(t9)
t10 = Transaction(
    Vijay,
    Ramesh.identity,
    3.0
)
t10.sign_transaction()
transactions.append(t10)

```

```

+ Code + Text | ⚙ Copy to Drive Connect ▾ | ✎ Editing | ^
[ ] for transaction in transactions:
    display_transaction (transaction)
    print ('-----')

    sender: 30819f300d06092a864886f70d010101050003818d00308189028181008f5e89df98216eb2c1e4713
    -----
    recipient: 30819f300d06092a864886f70d010101050003818d003081890281810082e40f44f9cef42cac49
    -----
    value: 15.0
    -----
    time: 2021-04-08 06:10:52.172517
    -----
    sender: 30819f300d06092a864886f70d010101050003818d00308189028181008f5e89df98216eb2c1e4713
    -----
    recipient: 30819f300d06092a864886f70d010101050003818d0030818902818100a0b9b51bb0c81cbbad86
    -----
    value: 6.0
    -----
    time: 2021-04-08 06:11:40.567785
    -----
    sender: 30819f300d06092a864886f70d010101050003818d003081890281810082e40f44f9cef42cac492d1
    -----
    recipient: 30819f300d06092a864886f70d010101050003818d0030818902818100d776af73071682cb494e
    -----
    value: 2.0
    -----
    time: 2021-04-08 06:11:40.569278
    -----
    sender: 30819f300d06092a864886f70d010101050003818d0030818902818100a0b9b51bb0c81cbbad86384
    -----
    recipient: 30819f300d06092a864886f70d010101050003818d003081890281810082e40f44f9cef42cac49
    -----
    value: 4.0
    -----
    time: 2021-04-08 06:11:40.570658
    -----
    sender: 30819f300d06092a864886f70d010101050003818d0030818902818100d776af73071682cb494e752
    -----
    recipient: 30819f300d06092a864886f70d010101050003818d0030818902818100a0b9b51bb0c81cbbad86
    -----
    value: 7.0
    -----
    time: 2021-04-08 06:11:40.572173
    -----
    sender: 30819f300d06092a864886f70d010101050003818d003081890281810082e40f44f9cef42cac492d1
    -----
    recipient: 30819f300d06092a864886f70d010101050003818d0030818902818100a0b9b51bb0c81cbbad86
    -----
    value: 3.0
    -----
    time: 2021-04-08 06:11:40.574026
[ ] class Block:
    def __init__(self):
        self.verified_transactions = []
        self.previous_block_hash = ""
        self.Nonce = ""

[ ] last_block_hash = ""

[ ] Dinesh = Client()

[ ] t0 = Transaction (
    "Genesis",
    Dinesh.identity,
    500.0
)

[ ] block0 = Block()

[ ] block0.previous_block_hash = None
    Nonce = None

[ ] block0.verified_transactions.append (t0)

```

```

+ Code + Text | ⚙ Copy to Drive Connect ▾ | ✎ Editing ▾

[ ] digest = hash(block0)
last_block_hash = digest

[ ] def dump_blockchain(self):
    print("Number of blocks in the chain: " + str(len(self)))
    for x in range(len(TPCoins)):
        block_temp = TPCoins[x]
        print("block # " + str(x))
        for transaction in block_temp.verified_transactions:
            display_transaction(transaction)
            print('-----')
        print('=====') 

[ ] TPCoins = []

[ ] def dump_blockchain(self):
    print("Number of blocks in the chain: " + str(len(self)))
    for x in range(len(TPCoins)):
        block_temp = TPCoins[x]
        print("block # " + str(x))
        for transaction in block_temp.verified_transactions:
            display_transaction(transaction)
            print('-----')
        print('=====') 

[ ] TPCoins.append(block0)

[ ] dump_blockchain(TPCoins)

Number of blocks in the chain: 1
block # 0
sender: Genesis
-----
recipient: 30819f300d06092a864886f70d010101050003818d0030818902818100bae5c5a46a1591af94c0
-----
value: 500.0
-----
time: 2021-04-08 06:16:28.464142
-----
-----
=====

[ ] def sha256(message):
    return hashlib.sha256(message.encode('ascii')).hexdigest()

[ ] def mine(message, difficulty=1):
    assert difficulty >= 1
    prefix = '1' * difficulty
    for i in range(1000):
        digest = sha256(str(hash(message)) + str(i))
        if digest.startswith(prefix):
            print("after " + str(i) + " iterations found nonce: " + digest)
            return digest

[ ] mine ("test message", 2)
'938c72d2197fa6c2294df7bc8cea6be98769aad888e3cfa15558c78469394ebd'

[ ] last_transaction_index = 0

[ ] block = Block()
for i in range(3):
    temp_transaction = transactions[last_transaction_index]
    # validate transaction
    # if valid
    block.verified_transactions.append(temp_transaction)
    last_transaction_index += 1

block.previous_block_hash = last_block_hash
block.Nonce = mine(block, 2)
digest = hash(block)
TPCoins.append(block)
last_block_hash = digest

```

+ Code + Text | Connect ▾ | Editing | ^

```
[ ] # Miner 2 adds a block
block = Block()

for i in range(3):
    temp_transaction = transactions[last_transaction_index]
    # validate transaction
    # if valid
    block.verified_transactions.append (temp_transaction)
    last_transaction_index += 1
block.previous_block_hash = last_block_hash
block.Nonce = mine(block, 2)
digest = hash (block)
TPCoins.append (block)
last_block_hash = digest
# Miner 3 adds a block
block = Block()

for i in range(3):
    temp_transaction = transactions[last_transaction_index]
    #display_transaction (temp_transaction)
    # validate transaction
    # if valid
    block.verified_transactions.append (temp_transaction)
    last_transaction_index += 1
block.previous_block_hash = last_block_hash
block.Nonce = mine (block, 2)
digest = hash (block)

TPCoins.append (block)
last_block_hash = digest

[ ] dump_blockchain(TPCoins)
time: 2021-04-08 06:11:40.570658
-----
-----
sender: 30819f300d06092a864886f70d010101050003818d0030818902818100d776af73071682cb494e752
-----
recipient: 30819f300d06092a864886f70d010101050003818d0030818902818100a0b9b51bb0c81cbbad86
-----
value: 7.0
-----
time: 2021-04-08 06:11:40.572173
-----
-----
sender: 30819f300d06092a864886f70d010101050003818d003081890281810082e40f44f9cef42cac492d1
-----
recipient: 30819f300d06092a864886f70d010101050003818d0030818902818100a0b9b51bb0c81cbbad86
-----
value: 3.0
-----
time: 2021-04-08 06:11:40.574036
-----
-----
block # 3
sender: 30819f300d06092a864886f70d010101050003818d0030818902818100a0b9b51bb0c81cbbad86384
-----
recipient: 30819f300d06092a864886f70d010101050003818d00308189028181008f5e89df98216eb2c1e4
-----
value: 8.0
-----
time: 2021-04-08 06:11:40.575310
-----
-----
sender: 30819f300d06092a864886f70d010101050003818d0030818902818100a0b9b51bb0c81cbbad86384
-----
recipient: 30819f300d06092a864886f70d010101050003818d003081890281810082e40f44f9cef42cac49
-----
value: 1.0
-----
time: 2021-04-08 06:11:40.576645
-----
-----
sender: 30819f300d06092a864886f70d010101050003818d0030818902818100d776af73071682cb494e752
-----
recipient: 30819f300d06092a864886f70d010101050003818d00308189028181008f5e89df98216eb2c1e4
-----
value: 5.0
-----
time: 2021-04-08 06:11:40.578007
-----
=====
```

Practical 4

Implement and demonstrate the use of the following in Solidity :

- (I).Varaible
- (II).Operators
- (III).Loops
- (IV).Decision Making
- (V).Strings

(I).Variable

```
pragma solidity ^0.5.0;

contract SolidityTest {

    uint storedData; // State variable

    constructor() public {
        storedData = 10;
    }

    function getResult() public view returns(uint){
        uint a = 1; // local variable
        uint b = 2;
        uint result = a + b;
        return storedData; //access the state variable
    }
}
```

Output:

The screenshot shows the Solidity Compiler interface. On the left, there are configuration options for the compiler (version 0.5.17+commit.d19bb13), language (Solidity), EVM version (compiler default), and compiler configuration (Auto compile, Enable optimization set to 200, Hide warnings). Below these are buttons for "Compile new one.sol" and "Contract". Under "Contract", it says "SolidityTest (new one.sol)". There are three publishing options: "Publish on Swarm" (with a file icon), "Publish on IPFS" (with a green checkmark icon), and "Compilation Details". At the bottom, there is a "Deploy" button and a note about publishing to IPFS.

```
pragma solidity ^0.5.0;
contract SolidityTest {
    uint storedData; // State variable
    constructor() public {
        storedData = 10;
    }
    function getResult() public view returns(uint){
        uint a = 1; // Local variable
        uint b = 2;
        uint result = a + b;
        return storedData; //access the state variable
    }
}
```

The screenshot shows the Deploy & Run Transactions interface. It has a "Deploy" section where "Value" is set to 0 wei, and a "Contract" dropdown showing "SolidityTest - new one.sol". Below this is a "Deploy" button. To the right, there is a "Transactions recorded" section showing a single entry for "SOLIDITYTEST AT 0xD91...39138 (MEM)". Under this, there is a "getResult" button and a "Low level interactions" section with a "CALLDATA" tab and a "Transact" button. The right side of the interface is identical to the Solidity Compiler screen, displaying the same Solidity code and transaction details.

```
pragma solidity ^0.5.0;
contract SolidityTest {
    uint storedData; // State variable
    constructor() public {
        storedData = 10;
    }
    function getResult() public view returns(uint){
        uint a = 1; // local variable
        uint b = 2;
        uint result = a + b;
        return storedData; //access the state variable
    }
}
```

```

// Solidity program to demonstrate state variables

pragma solidity ^0.5.0;

// Creating a contract

contract Solidity_var_Test {

// Declaring a state variable

    uint8 public state_var;

// Defining a constructor

constructor() public {

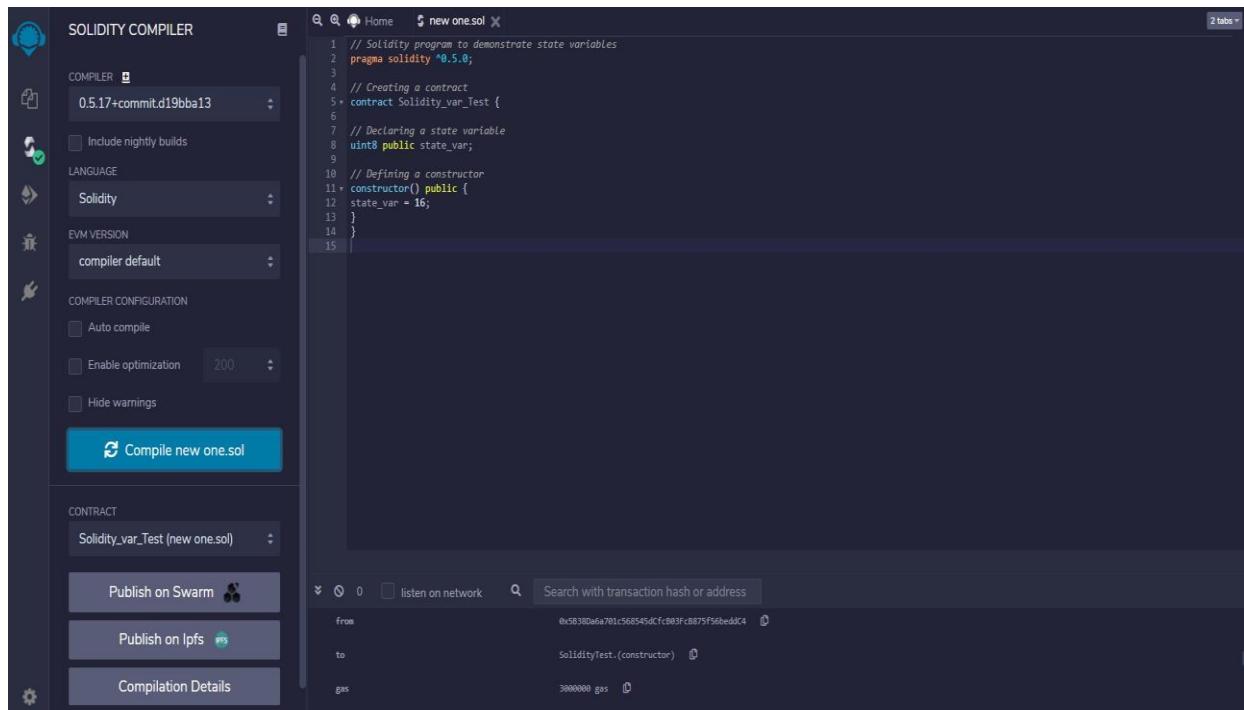
    state_var = 16;

}

}

```

Output:



The screenshot shows the Solidity Compiler interface with the following details:

- SOLIDITY COMPILER** section:
 - COMPILER: 0.5.17+commit.d19bba13
 - LANGUAGE: Solidity
 - EVM VERSION: compiler default
 - COMPILER CONFIGURATION: Auto compile, Enable optimization (set to 200), Hide warnings
 - Buttons: Compile new one.sol, Publish on Swarm, Publish on ipfs, Compilation Details
- CONTRACT** section: Solidity_var_Test (new one.sol)
- Output Area (new one.sol tab):**

```

1 // Solidity program to demonstrate state variables
2 pragma solidity ^0.5.0;
3
4 // Creating a contract
5+ contract Solidity_var_Test {
6
7 // Declaring a state variable
8     uint8 public state_var;
9
10 // Defining a constructor
11+ constructor() public {
12     state_var = 16;
13 }
14
15

```
- Deployment Area:**
 - from: 0x56380a6a701c568545dcfc083fc8875f50e6d04
 - to: SolidityTest.(constructor)
 - gas: 3000000 gas

The screenshot shows the Solidity Compiler interface. On the left, there's a sidebar with various icons. The main area has several dropdown menus and input fields:

- COMPILER:** Version 0.5.17+commit.d19bba13, with an option to include nightly builds.
- LANGUAGE:** Solidity.
- EVM VERSION:** compiler default.
- COMPILER CONFIGURATION:** Auto compile, Enable optimization (set to 200), Hide warnings.
- CONTRACT:** Solidity_var_Test (Variable.sol).

At the bottom, there are three buttons: Publish on Swarm, Publish on Ipfs, and Compilation Details. Below these are links for ABI and Bytecode.

The right side shows the source code of `Variable.sol`:

```
// Solidity program to demonstrate state variables
pragma solidity ^0.5.0;

// Creating a contract
contract Solidity_var_Test {

    // Declaring a state variable
    uint8 public state_var;

    // Defining a constructor
    constructor() public {
        state_var = 16;
    }
}
```

The screenshot shows the Truffle UI interface for deploying and running transactions. The top navigation bar includes icons for Home, Variable.sol, and a search function.

ENVIRONMENT: JavaScript VM

ACCOUNT: 0x5B3...eddC4 (99.99999999999999 Wei)

GAS LIMIT: 3000000

VALUE: 0 wei

CONTRACT: Solidity_var_Test - Variable.sol

Deploy button (orange)

Publish to IPFS

OR

At Address (selected) / Load contract from Address

Transactions recorded: 1

Deployed Contracts

SOLIDITY_VAR_TEST AT 0xD91...39138

state_var

0: uint8: 16

Low level interactions

CALldata

Transact button (orange)

```

// Solidity program to show Global variables

pragma solidity ^0.5.0;

// Creating a contract

contract Test {

    // Defining a variable

    address public admin;

    // Creating a constructor to

    // use Global variable

    constructor() public {

        admin = msg.sender;

    }

}

```

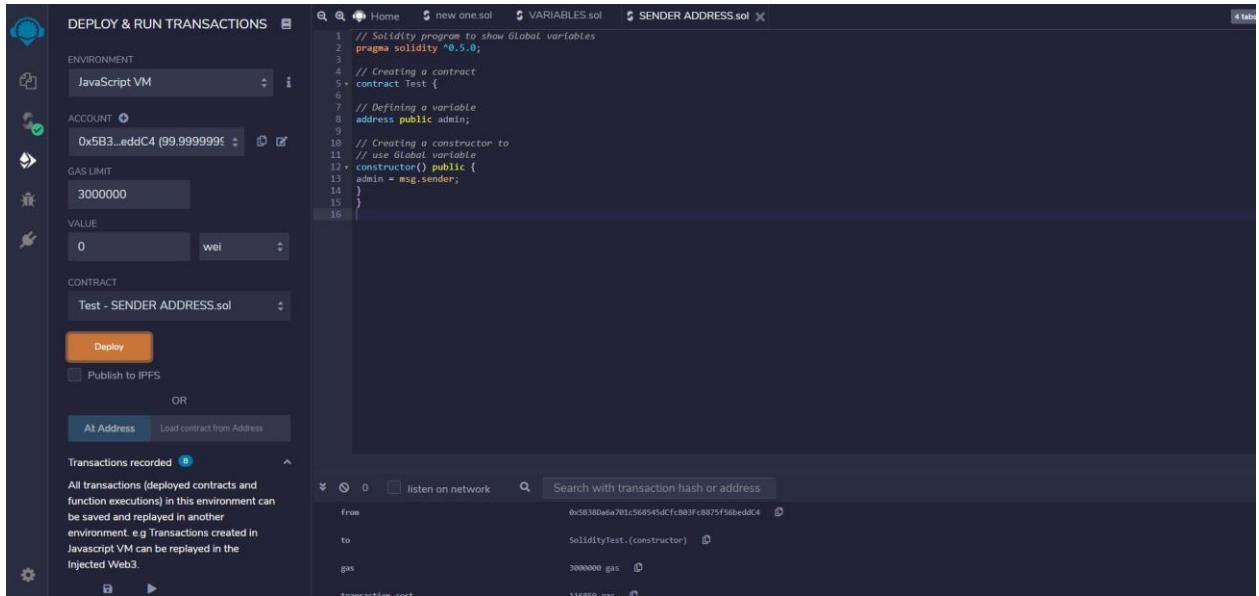
Output:

The screenshot shows the Truffle UI interface. On the left, the 'SOLIDITY COMPILER' sidebar is open, displaying compiler settings like version 0.5.17+commit.d19bba13, language Solidity, and EVM version compiler default. Below the sidebar, the 'CONTRACT' section shows 'Test (SENDER ADDRESS.sol)' selected. Underneath, there are buttons for 'Publish on Swarm' and 'Publish on Ipfs'. At the bottom of the sidebar, there's a 'Compilation Details' section. The main area of the screen displays the Solidity code. A large blue button at the bottom left of the main area says 'Compile SENDER ADDRESS.sol'. The code itself is identical to the one shown in the previous text block.

```

1 // Solidity program to show Global variables
2 pragma solidity ^0.5.0;
3
4 // Creating a contract
5 contract Test {
6
7     // Defining a variable
8     address public admin;
9
10 // Creating a constructor to
11 // use Global variable
12 constructor() public {
13     admin = msg.sender;
14 }
15
16

```



(II).Operators

```
// Solidity contract to demonstrate Arithematic Operator
```

```
pragma solidity ^0.5.0;
```

```
// Creating a contract
```

```
contract SolidityTest {
```

```
// Initializing variables
```

```
uint16 public a = 20;
```

```
uint16 public b = 10;
```

```
// Initializing a variable with sum
```

```
uint public sum = a + b;
```

```
// Initializing a variable with the difference
```

```
uint public diff = a - b;
```

```

// Initializing a variable with product
uint public mul = a * b;

// Initializing a variable with quotient
uint public div = a / b;

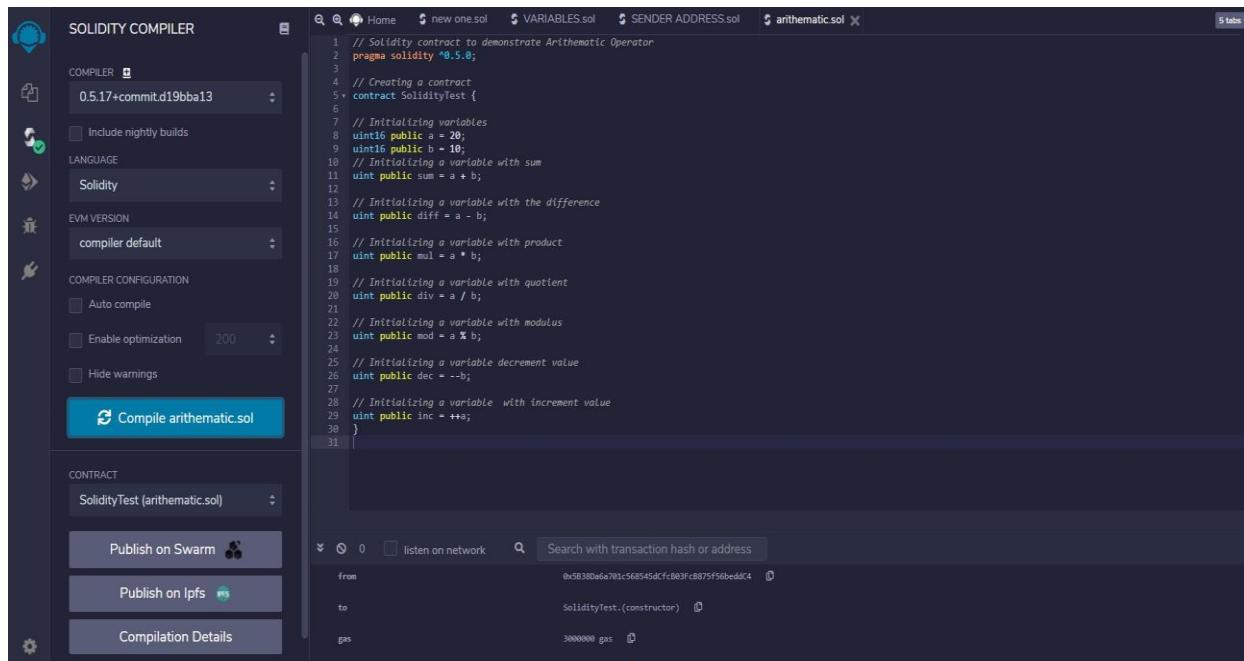
// Initializing a variable with modulus
uint public mod = a % b;

// Initializing a variable decrement value
uint public dec = --b;

// Initializing a variable with increment value
uint public inc = ++a;
}

```

Output:



The screenshot shows the Solidity Compiler interface with the following configuration:

- COMPILER:** 0.5.17+commit.d19bba13
- LANGUAGE:** Solidity
- EVM VERSION:** compiler default
- COMPILER CONFIGURATION:** Auto compile
- Compiler Optimization:** 200
- Warnings:** Hide warnings

The code area displays the Solidity code for `arithmetic.sol`:

```

1 // Solidity contract to demonstrate Arithmetic Operator
2 pragma solidity ^0.5.0;
3
4 // Creating a contract
5 contract SolidityTest {
6
7 // Initializing variables
8 uint16 public a = 20;
9 uint16 public b = 10;
10 // Initializing a variable with sum
11 uint public sum = a + b;
12
13 // Initializing a variable with the difference
14 uint public diff = a - b;
15
16 // Initializing a variable with product
17 uint public mul = a * b;
18
19 // Initializing a variable with quotient
20 uint public div = a / b;
21
22 // Initializing a variable with modulus
23 uint public mod = a % b;
24
25 // Initializing a variable decrement value
26 uint public dec = --b;
27
28 // Initializing a variable with increment value
29 uint public inc = ++a;
30 }
31

```

The bottom section shows the deployment details for the `SolidityTest` contract:

- Contract:** SolidityTest (`arithmetic.sol`)
- Publish Options:**
 - Published on Swarm
 - Published on IPFS
 - Compilation Details
- Deployment:**
 - From: 0x58380a6a701c568545dCfcB83FcB875f56beddC4
 - To: SolidityTest.(constructor)
 - Gas: 3000000 gas

DEPLOY & RUN TRANSACTIONS

ENVIRONMENT
JavaScript VM

ACCOUNT
0x5B3...eddC4 (99.999999)

GAS LIMIT
3000000

VALUE
0 wei

CONTRACT
SolidityTest - arithmetic.sol

Deploy

Publish to IPFS

OR

At Address **Load contract from Address**

Transactions recorded 9

All transactions (deployed contracts and function executions) in this environment can be saved and replayed in another environment. e.g Transactions created in Javascript VM can be replayed in Injected Web3.

from 0x5B3...eddC4

to SolidityTest.(constructor)

gas 3000000 gas

DEPLOY & RUN TRANSACTIONS

SOLIDITYTEST AT 0xD2A...FD005 (MEM)

a

b

dec

diff

div

inc

mod

mul

sum

Low level interactions

CALldata

Transact

from 0x5B3...eddC4

to SolidityTest.(constructor)

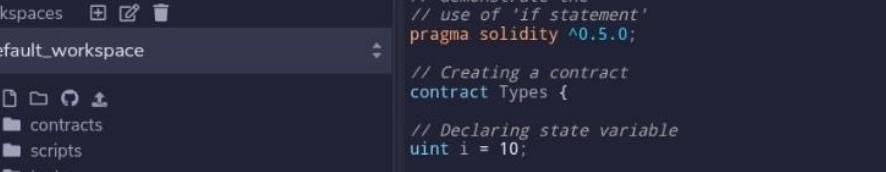
gas 3000000 gas

transaction cost 310000 gas

(IV).Decision Making

```
// Solidity program to demonstrate the use of 'if statement'  
pragma solidity ^0.5.0;  
  
// Creating a contract  
contract Types {  
  
    // Declaring state variable  
    uint i = 10;  
  
    // Defining function to demonstrate use of 'if statement'  
    function decision_making() public view returns(bool){  
        if(i<10){  
            return true;  
        }  
        }  
    }  
}
```

Output:



```
// Solidity program to
// demonstrate the
// use of 'if statement'
pragma solidity ^0.5.0;

// Creating a contract
contract Types {

    // Declaring state variable
    uint i = 10;

    // Defining function to
    // demonstrate use of
    // 'if statement'
    function decision_making() public view returns(bool){
        if(i<10){
            return true;
        }
    }
}
```

SOLIDITY COMPILER

COMPILER: 0.5.17+commit.d19bba13

Include nightly builds

LANGUAGE: Solidity

EVM VERSION: compiler default

COMPILER CONFIGURATION:

- Auto compile
- Enable optimization: 200
- Hide warnings

Compile Boolean.sol

CONTRACT: Types (Boolean.sol)

Publish on Swarm

Publish on Ipfs

Compilation Details

ABI Bytecode

Home Boolean.sol X

```
// Solidity program to
// demonstrate the
// use of 'if statement'
pragma solidity ^0.5.0;

// Creating a contract
contract Types {

    // Declaring state variable
    uint i = 10;

    // Defining function to
    // demonstrate use of
    // 'if statement'
    function decision_making() public view returns(bool){
        if(i<10){
            return true;
        }
    }
}
```

The screenshot shows the Truffle UI interface. On the left, there's a sidebar with icons for deploying, running, and monitoring transactions. The main area is divided into two sections: 'DEPLOY & RUN TRANSACTIONS' on the left and 'Boolean.sol' code editor on the right.

DEPLOY & RUN TRANSACTIONS

- ENVIRONMENT:** JavaScript VM
- ACCOUNT:** 0x5B3...eddC4 (99.99999999999999)
- GAS LIMIT:** 3000000
- VALUE:** 0 wei
- CONTRACT:** Types - Boolean.sol
- Buttons:** Deploy, Publish to IPFS
- OR:** At Address, Load contract from Address
- Transactions recorded:** 1
- Deployed Contracts:** TYPES AT 0xD91...3913B (MEMORY)
- Low level interactions:** CALLDATA, Transact

Boolean.sol

```
// Solidity program to demonstrate the use of 'if...else' statement
pragma solidity ^0.5.0;

// Creating a contract
contract Types {

    // Declaring state variables
    uint i = 10;

    // Defining function to demonstrate use of
    // 'if statement'
    function decision_making() public view returns(bool){
        if(i<10){
            return true;
        }
    }
}
```

```
// Solidity program to demonstrate the use of 'if...else' statement
```

```
pragma solidity ^0.5.0;
```

```
// Creating a contract
```

```
contract Types {
```

```
// Declaring state variables
```

```
uint i = 10;  
bool even;  
  
// Defining function to  
// demonstrate the use of  
// 'if...else statement'  
function decision_making() public {  
    if(i%2 == 0){  
        even = true;  
    }  
    else{  
        even = false;  
    }  
}  
  
function getresult() public view returns(bool)  
{  
    return even;  
}  
  
}
```

Output:

The screenshot shows a Solidity development environment. On the left is a sidebar with various icons for file management, including a workspace icon, a file icon, a folder icon, and a refresh icon. Below these are sections for contracts, scripts, tests, artifacts, and metadata files like Types_metadata.json and test_metadata.json. A README.txt file is also listed. The main area is a code editor with tabs for Boolean.sol, Book.sol, and If_else.sol. The If_else.sol tab is active, displaying the following Solidity code:

```
1 // Solidity program to
2 // demonstrate the use of
3 // 'if...else' statement
4 pragma solidity ^0.5.0;
5
6 // Creating a contract
7 contract Types {
8
9     // Declaring state variables
10    uint i = 10;
11    bool even;
12
13    // Defining function to
14    // demonstrate the use of
15    // 'if...else' statement
16    function decision_making() public {
17        if(i%2 == 0){
18            even = true;
19        }
20        else{
21            even = false;
22        }
23    }
24
25    function getresult() public view returns(bool)
26    {
27        return even;
28    }
29
30 }
```

The screenshot shows the Solidity Compiler interface. On the left, there's a sidebar with compiler settings: version 0.5.17+commit.d19bba13, language Solidity, EVM version compiler default, and compiler configuration options like auto compile, enable optimization (set to 200), and hide warnings. A prominent blue button labeled "Compile If_else.sol" is at the bottom of this sidebar. The main right-hand panel displays the same Solidity code as the previous screenshot. Below the code, there's a section titled "CONTRACT" showing "Types (If_else.sol)". Underneath are three buttons: "Publish on Swarm" (with a Swarm icon), "Publish on Ipfs" (with an IPFS icon), and "Compilation Details". At the very bottom of the interface are links for ABI and Bytecode.

The screenshot shows the Truffle UI interface for deploying and running Solidity transactions. On the left, there's a sidebar with icons for account management, gas limit, value, and contract deployment. The main area has tabs for 'Home', 'Boolean.sol', 'Book.sol', and 'If_else.sol'. The 'If_else.sol' tab is active, displaying the following Solidity code:

```
// Solidity program to demonstrate the use of 'if...else' statement
pragma solidity ^0.5.0;

contract Types {
    // Declaring state variables
    uint i = 10;
    bool even;

    // Defining function to demonstrate the use of 'if...else' statement
    function decision_making() public {
        if(i%2 == 0){
            even = true;
        } else{
            even = false;
        }
    }

    function getresult() public view returns(bool) {
        return even;
    }
}
```

The deployment section shows the contract 'Types - If_else.sol' selected, with a 'Deploy' button and a checkbox for 'Publish to IPFS'. Below this, there's an 'OR' section with tabs for 'At Address' and 'Load contract from Address'. Under 'Transactions recorded', there are three entries: 'TYPES AT 0xD91...39138 (MEMORY)', 'TEST AT 0xD8B...33FA8 (MEMORY)', and 'TYPES AT 0xDA0...42B53 (MEMORY)'. The third entry is expanded, showing two functions: 'decision_making...' and 'getresult'. The 'getresult' button is highlighted in blue. Below the functions, it says '0: bool: true'. At the bottom, there's a 'Low level interactions' section with a 'CALLDATA' input field and a 'Transact' button.

(III) Strings

```
// Solidity program to demonstrate
// how to create a contract
pragma solidity ^0.4.23;

// Creating a contract
contract Test {

    // Declaring variable
    string str;

    // Defining a constructor
    constructor(string str_in){
        str = str_in;
    }

    // Defining a function to
    // return value of variable 'str'
    function str_out() public view returns(string memory){
        return str;
    }
}
```

Output

The screenshot shows the Solidity Compiler interface. On the left, there are configuration options: Compiler (0.4.26+commit.4563c3fc), Language (Solidity), EVM Version (compiler default), Compiler Configuration (Auto compile, Enable optimization 200, Hide warnings), and a prominent blue button labeled "Compile string.sol". Below these are sections for CONTRACT (Test (string.sol)) and PUBLISH (Publish on Swarm, Publish on IPFS, Compilation Details). The main area displays the Solidity code:

```
1 // Solidity program to demonstrate
2 // how to create a contract
3 pragma solidity ^0.4.23;
4
5 // Creating a contract
6+ contract Test {
7
8 // Declaring variable
9 string str;
10
11 // Defining a constructor
12+ constructor(string str_in){
13 str = str_in;
14 }
15
16 // Defining a function to
17 // return value of variable 'str'
18+ function str_out() public view returns(string memory){
19 return str;
20 }
21 }
22
```

At the bottom, there are fields for from (0x5B38D6a701c568545dCf:003FcB875f56beddC4), to (SolidityTest.(constructor)), and gas (3000000 gas).

The screenshot shows the Deploy & Run Transactions interface. It features two tabs: "state_var" and "str_out". Both tabs have "Low level interactions" sections with "CALldata" inputs. The "state_var" tab has a "CALldata" input set to "0" with a "Transact" button. The "str_out" tab has a "CALldata" input with a "Transact" button. The main area displays the same Solidity code as the previous screenshot.

Below the tabs, there are fields for from (0x5B38D6a701c568545dCf:003FcB875f56beddC4), to (SolidityTest.(constructor)), and gas (3000000 gas).

Practical 5

Implement and demonstrate the use of the following in Solidity :

- (I).Arrays
- (II).Enums
- (III).Structs
- (IV).Mappings
- (V).Coversations
- (VI).Ether Units
- (VII).Special Varaibles

(I).Arrays

```
// Solidity program to demonstrate

// creating a fixed-size array

pragma solidity ^0.5.0;

// Creating a contract

contract Types {

    // Declaring state variables

    // of type array

    uint[6] data1;

    int[5] data;

    // Defining function to add

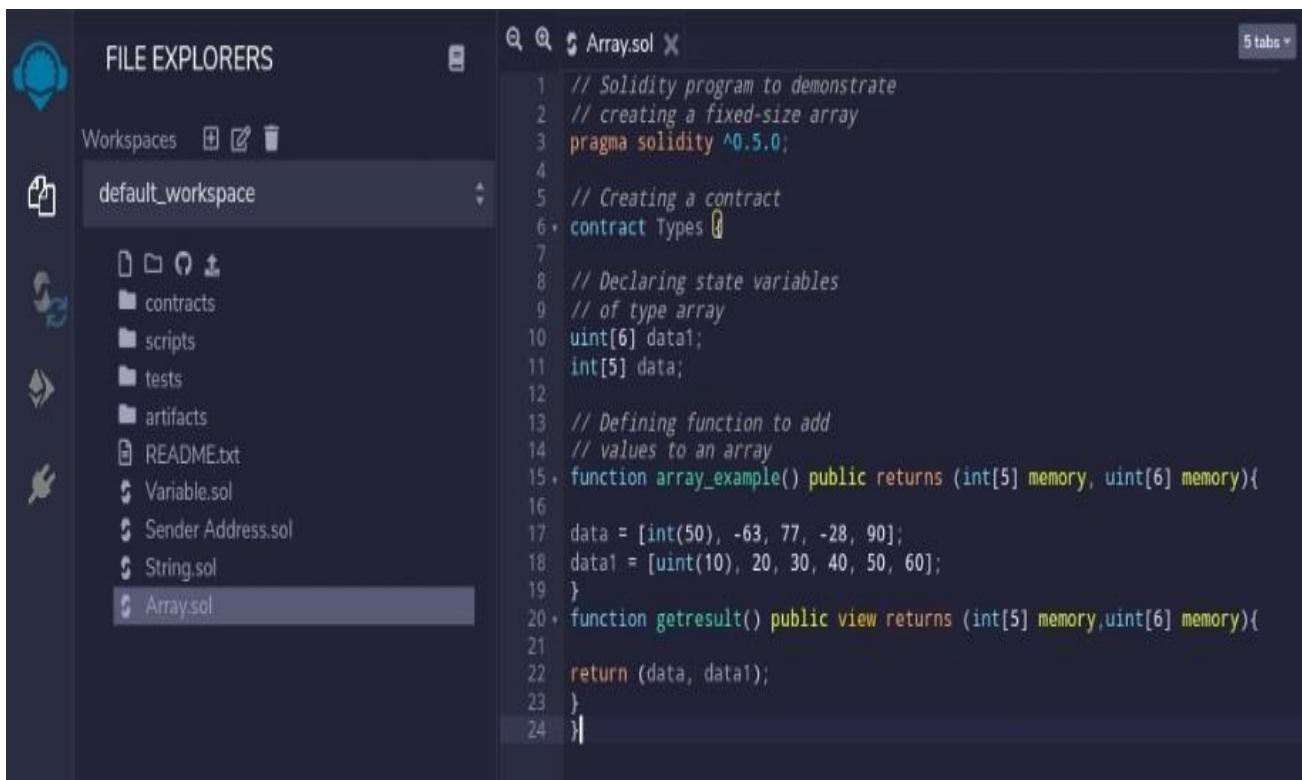
    // values to an array

    function array_example() public returns (int[5] memory, uint[6] memory){

        data = [int(50), -63, 77, -28, 90];
```

```
data1 = [uint(10), 20, 30, 40, 50, 60];  
}  
  
function getResult() public view returns (int[5] memory,uint[6] memory){  
return (data, data1);  
}  
}
```

Output:



The screenshot shows the Solidity IDE interface. On the left, the 'FILE EXPLORERS' panel displays a workspace named 'default_workspace' containing files: contracts, scripts, tests, artifacts, README.txt, Variable.sol, SenderAddress.sol, String.sol, and Array.sol. The 'Array.sol' file is currently selected and highlighted with a blue bar at the bottom. On the right, the main code editor panel shows the Solidity code for 'Array.sol'. The code defines a contract 'Types' with state variables 'data1' and 'data', and a function 'array_example()' that initializes 'data' and returns both 'data' and 'data1' from the 'getResult()' function.

```
// Solidity program to demonstrate  
// creating a fixed-size array  
pragma solidity ^0.5.0;  
  
// Creating a contract  
contract Types {  
  
    // Declaring state variables  
    // of type array  
    uint[6] data1;  
    int[5] data;  
  
    // Defining function to add  
    // values to an array  
    function array_example() public returns (int[5] memory, uint[6] memory){  
        data = [int(50), -63, 77, -28, 90];  
        data1 = [uint(10), 20, 30, 40, 50, 60];  
    }  
    function getResult() public view returns (int[5] memory,uint[6] memory){  
        return (data, data1);  
    }  
}
```

The screenshot shows the Solidity Compiler interface. On the left, there's a sidebar with various icons. The main area has tabs for "SOLIDITY COMPILER" and "ARRAY". The "SOLIDITY COMPILER" tab is active, showing the following configuration:

- COMPILER:** 0.5.17+commit.d19bba13
- Include nightly builds
- LANGUAGE:** Solidity
- EVM VERSION:** compiler default
- COMPILER CONFIGURATION:**
 - Auto compile
 - Enable optimization (set to 200)
 - Hide warnings

A large blue button at the bottom left says "Compile Array.sol".

In the main pane, titled "Array.sol", the Solidity code is displayed:

```
// Solidity program to demonstrate
// creating a fixed-size array
pragma solidity ^0.5.0;

// Creating a contract
contract Types {

    // Declaring state variables
    // of type array
    uint[6] data1;
    int[5] data;

    // Defining function to add
    // values to an array
    function array_example() public returns (int[5] memory, uint[6] memory){
        data = [int(50), -63, 77, -28, 90];
        data1 = [uint(10), 20, 30, 40, 50, 60];
    }

    function getResult() public view returns (int[5] memory,uint[6] memory){
        return (data, data1);
    }
}
```

Below the code, under the "CONTRACT" section, it says "Types (Array.sol)". There are three buttons:

- Publish on Swarm (with a Swarm icon)
- Publish on Ipfs (with an IPFS icon)
- Compilation Details

At the bottom, there are links for ABI and Bytecode.

```

1 // Solidity program to demonstrate
2 // creating a fixed-size array
3 pragma solidity ^0.5.0;
4
5 // Creating a contract
6 contract Types {
7
8 // Declaring state variables
9 // of type array
10 uint[6] data1;
11 int[5] data;
12
13 // Defining function to add
14 // values to an array
15 function array_example() public returns (int[5] memory, uint[6] memory){
16
17 data = [int(50), -63, 77, -28, 90];
18 data1 = [uint(10), 20, 30, 40, 50, 60];
19 }
20 function getresult() public view returns (int[5] memory,uint[6] memory){
21
22 return (data, data1);
23 }
24 }

```

(III).Structs

```

pragma solidity ^0.5.0;

contract test {

struct Book {
    string title;
    string author;
    uint book_id;
}

Book book;

```

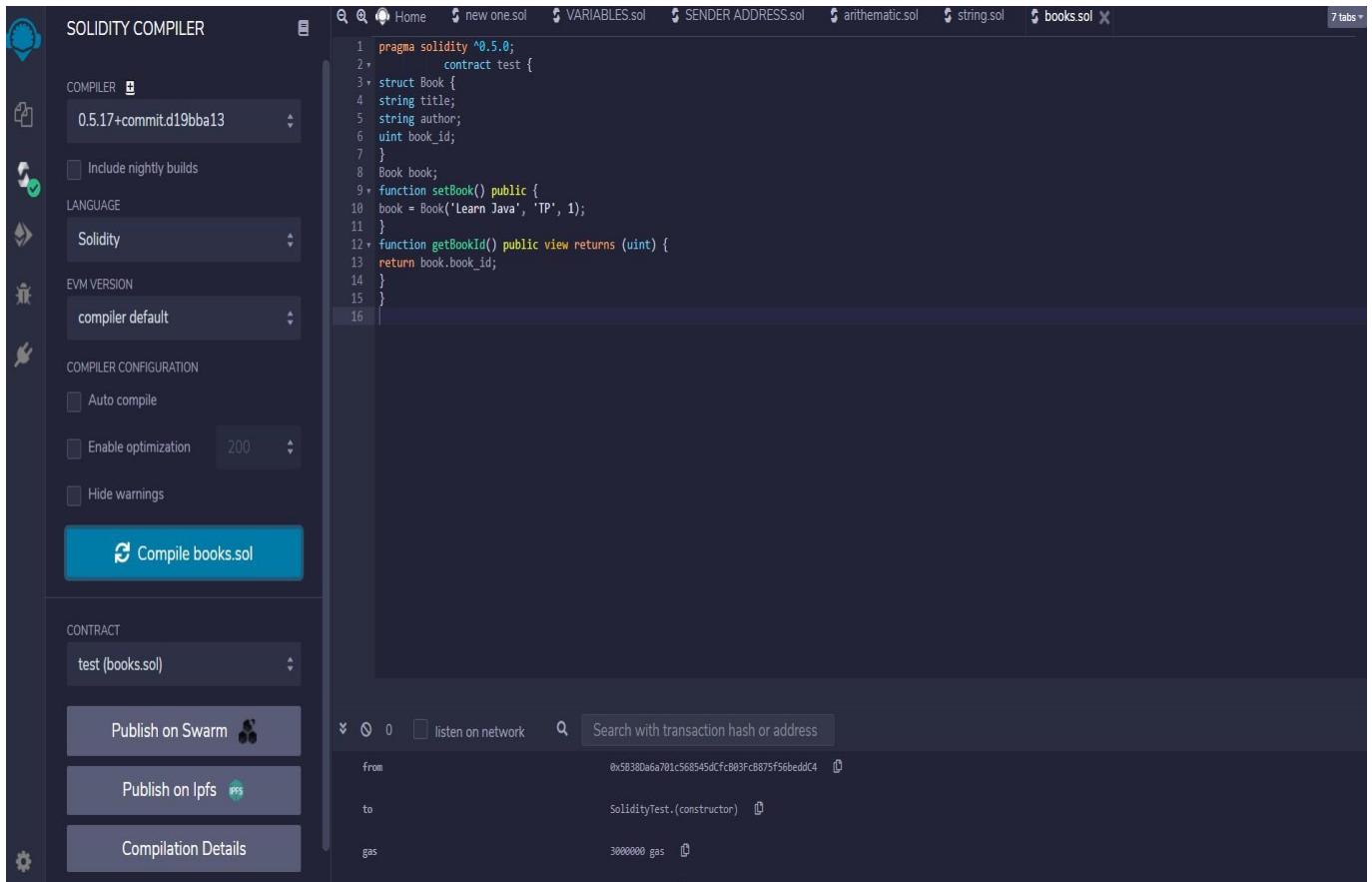
```

function setBook() public {
    book = Book('Learn Java', 'TP', 1);
}

function getBookId() public view returns (uint) {
    return book.book_id;
}

```

Output:



The screenshot shows the Solidity Compiler interface with the following details:

- Compiler:** 0.5.17+commit.d19bba13
- Language:** Solidity
- EVM Version:** compiler default
- Compiler Configuration:**
 - Auto compile
 - Enable optimization: 200
 - Hide warnings
- Contract:** test (books.sol)
- Buttons:**
 - Compile books.sol (highlighted in blue)
 - Publish on Swarm
 - Publish on IPFS
 - Compilation Details
- Compiled Output (Left Panel):**

```

1 pragma solidity ^0.5.0;
2 contract test {
3     struct Book {
4         string title;
5         string author;
6         uint book_id;
7     }
8     Book book;
9     function setBook() public {
10        book = Book('Learn Java', 'TP', 1);
11    }
12    function getBookId() public view returns (uint) {
13        return book.book_id;
14    }
15 }
16

```
- Transaction Details (Right Panel):**
 - from: 0x5B38D0a701c568545dCfcB03Fc8875f56beddC4
 - to: SolidityTest.(constructor)
 - gas: 3000000

The screenshot shows the Truffle IDE interface. On the left, there's a sidebar with icons for file operations like Open, Save, and Delete. Below that is a section for "DEPLOY & RUN TRANSACTIONS" which lists three test environments: "TEST AT 0x9D7...B5E99 (MEMORY)", "SOLIDITYTEST AT 0xD2A...FD005 (MEMORY)", and "TEST AT 0xDDA...5482D (MEMORY)". Each environment has a "str_out" button. Below this is a "Low level interactions" section for "CALLDATA" with a "Transact" button. Another "TEST AT 0x0FC...9AB36 (MEMORY)" section is also present with its own "setBook" and "getBookId" buttons and a "Transact" button. At the bottom, there's a transaction details panel with fields for "from" (0x58380a6a701c5685450cfcb083fc8875f56bed4c4), "to" (SolidityTest.(constructor)), "gas" (3000000), and "transaction cost" (116859 gas). A search bar at the top right says "Search with transaction hash or address".

```

1 pragma solidity ^0.5.0;
2+   contract test {
3+     struct Book {
4       string title;
5       string author;
6       uint book_id;
7     }
8     Book book;
9     function setBook() public {
10    book = Book("Learn Java", "IP", 1);
11  }
12+   function getBookId() public view returns (uint) {
13    return book.book_id;
14  }
15 }
16

```

(IV).Mappings

```

pragma solidity ^0.5.0;

contract LedgerBalance {
  mapping(address => uint) balance;

  function updateBalance() public returns(uint) {
    balance[msg.sender]=20;
    return balance[msg.sender];
  }
}

```

Output:

The image shows the Solidity Compiler interface. On the left, there's a sidebar with various icons: a blue headphones icon, a white square with a blue border, a green circular icon with a checkmark, a blue diamond icon, and a hand icon. The main area is titled "SOLIDITY COMPILER".

COMPILER: 0.5.17+commit.d19bba13

Include nightly builds

LANGUAGE: Solidity

EVM VERSION: compiler default

COMPILER CONFIGURATION:

- Auto compile
- Enable optimization 200
- Hide warnings

Compile Mapping.sol (button)

CONTRACT: LedgerBalance (Mapping.sol)

Publish on Swarm (with a Swarm icon)

Publish on Ipfs (with an IPFS icon)

Compilation Details

ABI Bytecode

On the right, there are two tabs: "Function_program.sol" and "Mapping.sol". The "Mapping.sol" tab is active, showing the following Solidity code:

```
pragma solidity ^0.5.0;
contract LedgerBalance {
    mapping(address => uint) balance;
    function updateBalance() public returns(uint)
    balance[msg.sender]=20;
    return balance[msg.sender];
}
```

The screenshot shows the Truffle UI interface for deploying and running Solidity contracts. On the left, there's a sidebar with icons for deploying, running, and monitoring. The main area is divided into sections:

- DEPLOY & RUN TRANSACTIONS**:
 - ENVIRONMENT**: JavaScript VM
 - ACCOUNT**: 0x5B3...eddC4 (99.99999999999999)
 - GAS LIMIT**: 3000000
 - VALUE**: 0 wei
 - CONTRACT**: LedgerBalance - Mapping.sol
 - Deploy** button
 - Publish to IPFS
- Transactions recorded**: 15
- Deployed Contracts**:
 - > TYPES AT 0XD91...39138 (MEMORY)
 - > TEST AT 0XD8B...33FA8 (MEMORY)
 - > TYPES AT 0XDA0...42B53 (MEMORY)
 - > TEST AT 0X9D7...B5E99 (MEMORY)
 - > LEDGERBALANCE AT 0XD2A...FD005 (I)
- Low level interactions**:
 - CALldata
 - updateBalance** button
- Transact** button

```

pragma solidity ^0.5.0;

contract LedgerBalance {

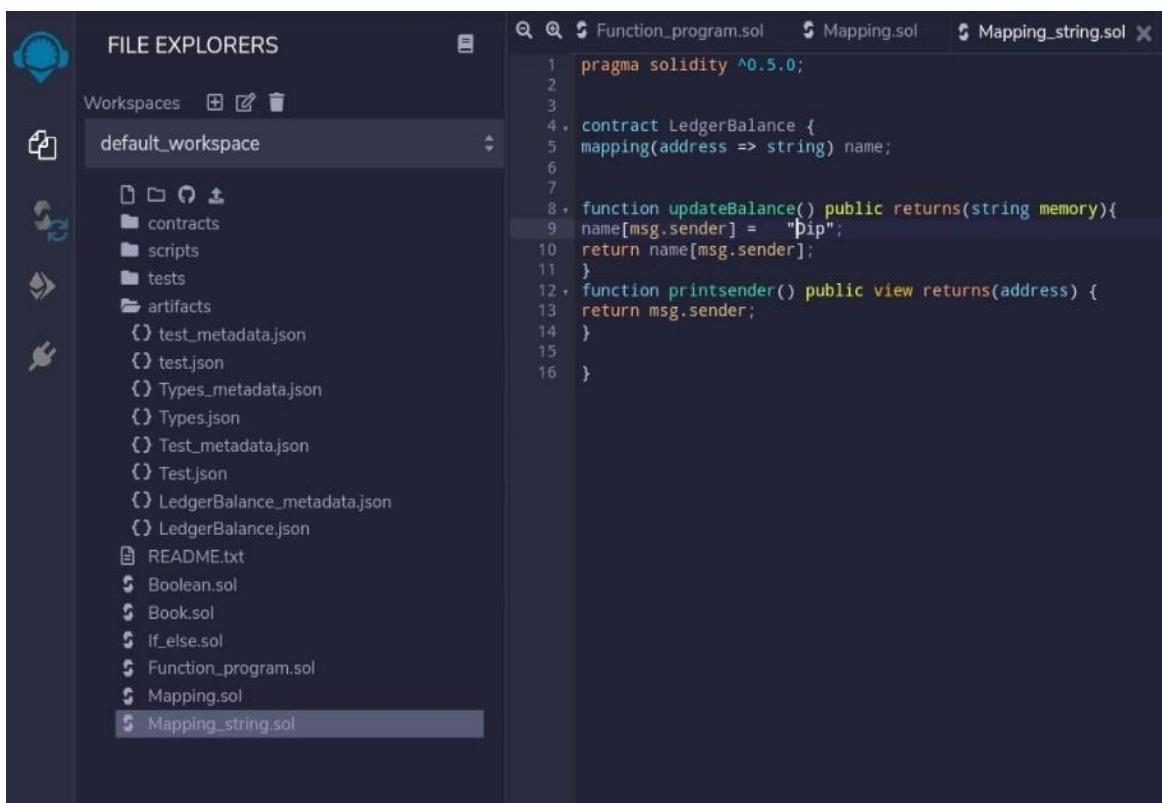
mapping(address => string) name;

function updateBalance() public returns(string memory){
name[msg.sender] = "Dip";
return name[msg.sender];
}

function printsender() public view returns(address) {
return msg.sender;
}
}

```

Output:



The screenshot shows the Solidity IDE interface. On the left, the 'FILE EXPLORERS' panel displays a file tree under 'default_workspace'. It includes folders for contracts, scripts, tests, and artifacts, along with JSON files like test_metadata.json, test.json, and Types.json. Several Solidity source files are listed at the bottom: README.txt, Boolean.sol, Book.sol, If_else.sol, Function_program.sol, Mapping.sol, and Mapping_string.sol. The 'Function_program.sol' file is currently open in the main code editor window, showing the provided Solidity code. The code editor has tabs for Function_program.sol, Mapping.sol, and Mapping_string.sol.

```

1 pragma solidity ^0.5.0;
2
3
4 contract LedgerBalance {
5 mapping(address => string) name;
6
7
8 function updateBalance() public returns(string memory){
9 name[msg.sender] = "Dip";
10 return name[msg.sender];
11 }
12 function printsender() public view returns(address) {
13 return msg.sender;
14 }
15
16 }

```

The Solidity Compiler interface shows the following configuration:

- COMPILER:** 0.5.17+commit.d19bba13
- LANGUAGE:** Solidity
- EVM VERSION:** compiler default
- COMPILER CONFIGURATION:**
 - Auto compile
 - Enable optimization: 200
 - Hide warnings
- Contract:** LedgerBalance (Mapping_string.sol)
- Buttons:** Publish on Swarm, Publish on Ipfs, Compilation Details
- ABI and Bytecode:** Links to ABI and Bytecode details.

The right panel displays the Solidity source code for `Mapping_string.sol`:

```
pragma solidity ^0.5.0;

contract LedgerBalance {
    mapping(address => string) name;

    function updateBalance() public returns(string memory){
        name[msg.sender] = "Dip";
        return name[msg.sender];
    }
    function printsender() public view returns(address) {
        return msg.sender;
    }
}
```

The interface allows testing interactions with the `LedgerBalance` contract at address `0x5B38Da6a701c568545dCfcB03FcB875f56beddC4`.

Test cases listed:

- TEST AT 0XD8B..33FA8 (MEMORY)
- TYPES AT 0XDA0..42B53 (MEMORY)
- TEST AT 0X9D7..B5E99 (MEMORY)
- LEDERBALANCE AT 0XD2A..FD005 (I)
- LEDERBALANCE AT 0X332..D4B6D (I)

Interaction buttons:

- updateBalance
- printsender

Low level interactions:

- CALldata
- Transact

Practical 6

Implement and demonstrate the use of the following in Solidity :

- (I).Functions
- (II).View Functions
- (III).Pure Functions
- (IV).Fallback Functions
- (V).Function Overloading
- (VI).Mathematical Functions
- (VII).Cryptographic Functions

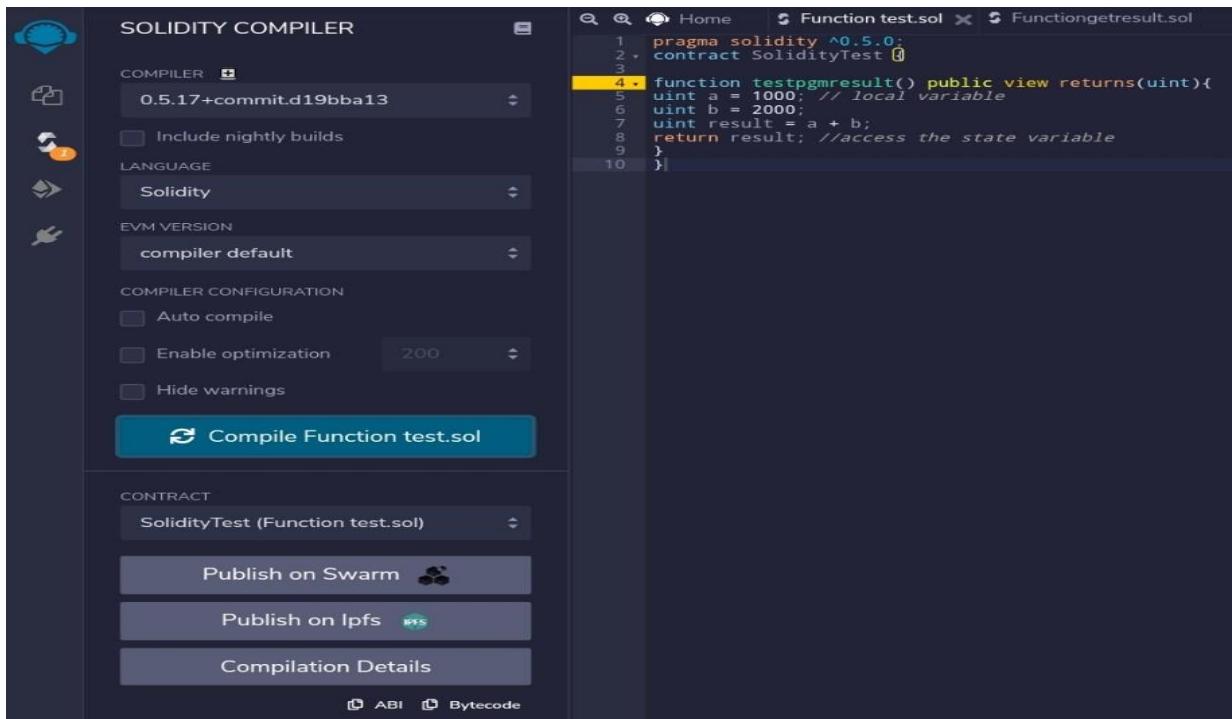
(I).Functions

```
pragma solidity ^0.5.0;

contract SolidityTest {

    function testpgmresult() public view returns(uint){
        uint a = 1000; // local variable
        uint b = 2000;
        uint result = a + b;
        return result; //access the state variable
    }
}
```

Output:



The image shows the Solidity Compiler interface. On the left, there's a sidebar with various icons and settings. The main area is a code editor with tabs for "Function test.sol" and "Functiongetresult.sol". The "test.sol" tab contains the following Solidity code:

```
pragma solidity ^0.5.0;
contract SolidityTest {
    function test() public view returns(uint) {
        uint a = 1000; // local variable
        uint b = 2000;
        uint result = a + b;
        return result; //access the state variable
    }
}
```

The sidebar includes sections for Compiler (version 0.5.17+commit.d19bba13), Language (Solidity), EVM Version (compiler default), Compiler Configuration (Auto compile, Enable optimization set to 200, Hide warnings), and CONTRACT (SolidityTest (Function test.sol)). Buttons for "Publish on Swarm" and "Publish on Ipfs" are also present.

The screenshot shows the Truffle UI interface. On the left, there's a sidebar with icons for deploying, running, and monitoring transactions. The main area is titled "DEPLOY & RUN TRANSACTIONS". It includes fields for "ENVIRONMENT" (set to "JavaScript VM"), "ACCOUNT" (set to "0x5B3...eddC4 (99.999999ε)"), "GAS LIMIT" (set to "3000000"), and "VALUE" (set to "0 wei"). Under "CONTRACT", it shows "SolidityTest - Function test.sol" and has a "Deploy" button. There's also a checkbox for "Publish to IPFS". Below that, it says "OR" and has tabs for "At Address" (selected) and "Load contract from Address". A dropdown menu shows "Transactions recorded" (2) and "Deployed Contracts". Under "Deployed Contracts", there are two entries: "SOLIDITYTEST AT 0xD91...39138 (MEM)" and "SOLIDITYTEST AT 0xD8B...33FAB (MEM)". The first entry is expanded, showing a function call "testpgmresult" with a result of "0: uint256: 3000". Below this, there's a section for "Low level interactions" with a "CALLDATA" tab and a "Transact" button.

```
1 pragma solidity ^0.5.0;
2 contract SolidityTest {
3     function testpgmresult() public view returns(uint){
4         uint a = 1000; // local variable
5         uint b = 2000;
6         uint result = a + b;
7         return result; //access the state variable
8     }
9 }
```

(II).View Functions

```
pragma solidity ^0.5.0;

contract Test {

    function getResult() public view returns(uint product, uint sum){

        uint a = 1; // local variable

        uint b = 2;

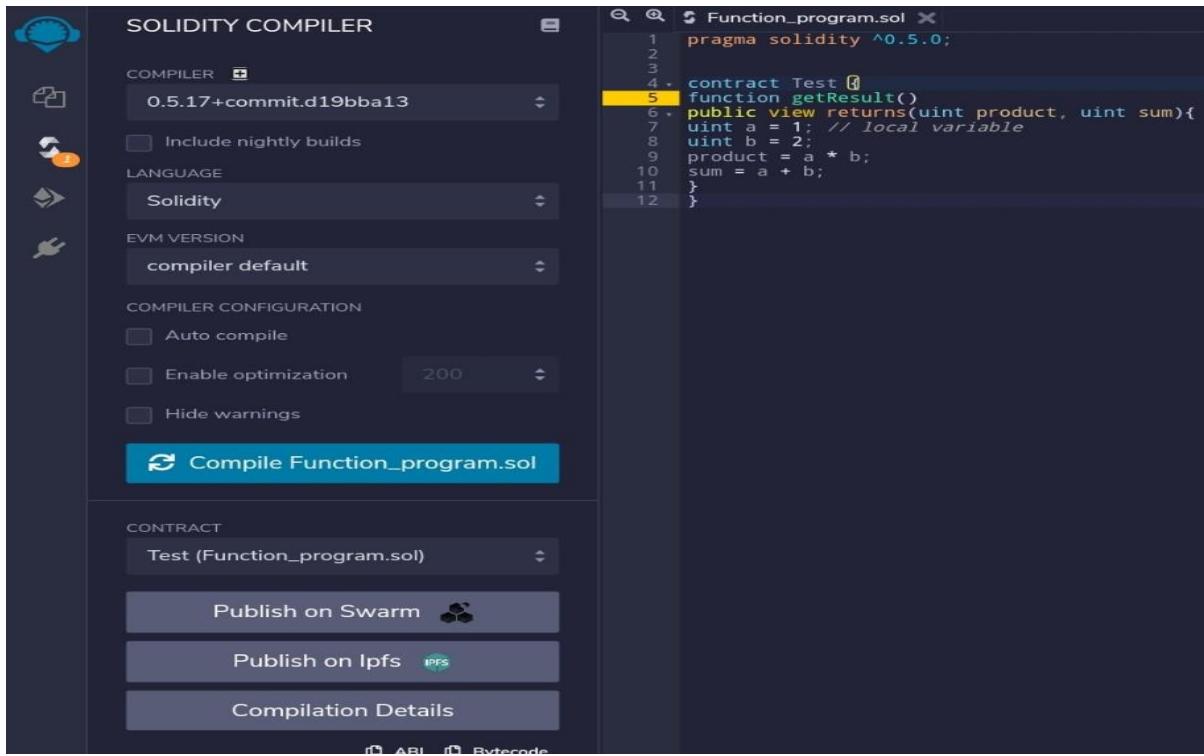
        product = a * b;

        sum = a + b;

    }

}
```

Output:



The image shows the Solidity Compiler interface. On the left, there's a sidebar with icons for Ethereum, Truffle, and Hardhat. The main area is titled "SOLIDITY COMPILER". It includes a "COMPILER" dropdown set to "0.5.17+commit.d19bba13", a "Include nightly builds" checkbox, a "LANGUAGE" dropdown set to "Solidity", an "EVM VERSION" dropdown set to "compiler default", and a "COMPILER CONFIGURATION" section with checkboxes for "Auto compile", "Enable optimization" (set to 200), and "Hide warnings". A prominent blue button at the bottom left says "Compile Function_program.sol". To the right, a code editor window titled "Function_program.sol" shows the following Solidity code:

```
pragma solidity ^0.5.0;

contract Test {
    function getResult() public view returns(uint product, uint sum){
        uint a = 1; // local variable
        uint b = 2;
        product = a * b;
        sum = a + b;
    }
}
```

```

DEPLOY & RUN TRANSACTIONS
ENVIRONMENT
JavaScript VM
ACCOUNT
0x5B3...eddC4 (99.9999995)
GAS LIMIT
3000000
VALUE
0 wei
CONTRACT
Test - Function_program.sol
Deploy
Publish to IPFS
OR
At Address Load contract from Address
Transactions recorded 8
Deployed Contracts
▶ TYPES AT 0XD91...39138 (MEMORY) ⏪ ×
▶ TEST AT 0XDBB...33FA8 (MEMORY) ⏪ ×
▶ TYPES AT 0XA0...42B53 (MEMORY) ⏪ ×
▼ TEST AT 0X9D7...B5E99 (MEMORY) ⏪ ×
getResult
0: uint256: product 2
1: uint256: sum 3

```

(III).Pure Functions

```

pragma solidity ^0.5.0;

contract C {
    //private state variable
    uint private data;

    //public state variable
    uint public info;

    //constructor

```

```

constructor() public {
    info = 10;
}

//private function

function increment(uint a) private pure returns(uint) { return a + 1; }

//public function

function updateData(uint a) public { data = a; }

function getData() public view returns(uint) { return data; }

function compute(uint a, uint b) internal pure returns (uint) { return a + b; }

}

//Derived Contract

contract E is C {
    uint private result;
    C private c;

    constructor() public {
        c = new C();
    }

    function getComputedResult() public {
        result = compute(3, 5);
    }

    function getResult() public view returns(uint) { return result; }

    function getData() public view returns(uint) { return c.info(); }

}

```

Output:

SOLIDITY COMPILER

COMPILER: 0.5.17+commit.d19bba13

LANGUAGE: Solidity

EVM VERSION: compiler default

COMPILER CONFIGURATION: Auto compile, Enable optimization (200), Hide warnings

Compile calling function external.sol

CONTRACT: C (calling function external.sol)

Publish on Swarm, Publish on ipfs

Compilation Details

DEPLOY & RUN TRANSACTIONS

TEST AT 0x0FC...9A836 (MEMORY)

- setBook: uint256 1
- getBookId: uint256 1

Low level interactions

CALLED DATA

CALLDATA: **Transact**

C AT 0XAEO...96B8B (MEMORY)

- updateData: 25
- getData: uint256 25
- info: uint256 10

Transactions

from: 0x58380a6a701c568545dCfcB03FcB875f56beddC4
to: SolidityTest.(constructor)
gas: 3000000 gas

Search with transaction hash or address

```

1 pragma solidity ^0.5.0;
2 contract C {
3     //private state variable
4     uint private data;
5
6     //public state variable
7     uint public info;
8
9     //constructor
10    constructor() public {
11        info = 10;
12    }
13    //private function
14    function increment(uint a) private pure returns(uint) { return a + 1; }
15
16    //public function
17    function updateData(uint a) public { data = a; }
18    function getData() public view returns(uint) { return data; }
19    function compute(uint a, uint b) internal pure returns (uint) { return a + b; }
20 }
21
22 //Derived Contract
23 contract E is C {
24     uint private result;
25     C private c;
26
27    constructor() public {
28        c = new C();
29    }
30    function getComputedResult() public {
31        result = compute(3, 5);
32    }
33    function getResult() public view returns(uint) { return result; }
34    function getData() public view returns(uint) { return c.info(); }
35 }
36

```

from: 0x58380a6a701c568545dCfcB03FcB875f56beddC4
to: SolidityTest.(constructor)
gas: 3000000 gas

Search with transaction hash or address

(V).Function Overloading

```
pragma solidity ^0.5.0;

contract Test {

function getSum(uint a, uint b) public pure returns(uint){
return a + b;
}

function getSum(uint a, uint b, uint c) public pure returns(uint){
return a + b + c;
}

function callSumWithTwoArguments() public pure returns(uint){
return getSum(1,2);
}

function callSumWithThreeArguments() public pure returns(uint){
return getSum(1,2,3);
}
}
```

Output:

The screenshot shows the Solidity Compiler interface with the following configuration:

- COMPILER:** 0.5.17+commit.d19bba13
- LANGUAGE:** Solidity
- EVM VERSION:** compiler default
- COMPILER CONFIGURATION:** Auto compile, Enable optimization (200), Hide warnings
- CONTRACT:** Test (overloading.sol)
- Publish Options:** Publish on Swarm, Publish on Ipfs
- Compilation Details:** Shows the transaction details for the deployment of the contract.

The code editor on the right contains the Solidity code for function overloading, which is identical to the code provided in the question.

```

1 pragma solidity ^0.5.0;
2 contract Test {
3     function getSum(uint a, uint b) public pure returns(uint){
4         return a + b;
5     }
6     function getSum(uint a, uint b, uint c) public pure returns(uint){
7         return a + b + c;
8     }
9     function callSumWithTwoArguments() public pure returns(uint){
10        return getSum(1,2);
11    }
12    function callSumWithThreeArguments() public pure returns(uint){
13        return getSum(1,2,3);
14    }
15 }
16

```

getSum
a: 3
b: 4
call
0: uint256: 7

getSum
a: 2
b: 3
c: 4
call

Search with transaction hash or address
from: 0x5B38D6a701568545dCfC803Fcb875f56e6dE4
to: SolidityTest.(constructor)
gas: 3000000 gas
transaction cost: 116859 gas

(VI).Mathematical Functions

```

pragma solidity ^0.5.0;

contract Test {

    function callAddMod() public pure returns(uint){
        return addmod(4, 5, 3);
    }

    function callMulMod() public pure returns(uint){
        return mulmod(4, 5, 3);
    }
}

```

Output:

SOLIDITY COMPILER

COMPILER: 0.5.17+commit.d19bba13
Include nightly builds

LANGUAGE: Solidity
EVM VERSION: compiler default

COMPILER CONFIGURATION:
Auto compile
Enable optimization: 200
Hide warnings

Compile Calladdmod.sol

CONTRACT: Test (Calladdmod.sol)
Publish on Swarm
Publish on Ipfs
Compilation Details
ABI
Bytecode

The screenshot shows the Solidity Compiler interface. On the left, there's a sidebar with icons for file operations, a search bar, and tabs for Home, Overloading.sol, and Calladdmod.sol. The main area has sections for Compiler (version 0.5.17), Language (Solidity), and EVM Version (compiler default). Compiler configuration includes Auto compile, Enable optimization (set to 200), and Hide warnings. Below these are buttons for Compile (with a circular arrow icon) and Publish on Swarm, Publish on Ipfs, and Compilation Details. At the bottom are links for ABI and Bytecode. The right side shows the Solidity code for the Test contract, which includes two functions: callAddMod() and callMulMod().

Contract

CONTRACT: Test - Calladdmod.sol
Deploy
Publish to IPFS

OR

At Address
Load contract from Address
Transactions recorded (2)
Deployed Contracts
TEST AT 0xD91...39138 (MEMORY)
TEST AT 0xD8B...33FAB (MEMORY)

callAddMod
0: uint256: 0

callMulMod
0: uint256: 2

Low level interactions
CALldata
Transact

The screenshot shows the Contract interface. It displays a list of deployed contracts under 'Deployed Contracts'. Two contracts are listed: 'TEST AT 0xD91...39138 (MEMORY)' and 'TEST AT 0xD8B...33FAB (MEMORY)'. Under each contract, there are buttons for 'callAddMod' and 'callMulMod'. The 'callAddMod' button for the first contract has a value of '0: uint256: 0'. The 'callMulMod' button for the second contract has a value of '0: uint256: 2'. At the bottom, there's a section for 'Low level interactions' with a 'CALldata' input field and a 'Transact' button.

(VII).Cryptographic Functions

```
pragma solidity ^0.5.0;

contract Test {

function callsha256() public pure returns( bytes32 result){

return sha256("ronaldo");

}

function callkeccak256() public pure returns( bytes32 result){

return keccak256("ronaldo");

}

}
```

Output:

The screenshot shows the Truffle UI interface. On the left, the 'SOLIDITY COMPILER' sidebar is visible, displaying compiler version 0.5.17+commit.d19bba13, language Solidity, EVM version compiler default, and compiler configuration options like Auto compile, Enable optimization (set to 200), and Hide warnings. A prominent blue button labeled 'Compile cryptodata.sol' is at the bottom of this sidebar. To the right, the main workspace displays two tabs: 'overloading.sol' and 'cryptodata.sol'. The 'cryptodata.sol' tab contains the Solidity code provided above. Below the tabs, there's a search bar and a transaction details panel. The transaction details show a transaction from address 0x58380da701c968545dCfc980f8075f56beddC4 to contract SolidityTest.(constructor) with a gas limit of 3000000 gas.

The screenshot shows the Truffle UI interface for deploying and running transactions. On the left, there's a sidebar with various icons. The main area has tabs for "overloading.sol" and "cryptodata.sol".

overloading.sol:

```
1 pragma solidity ^0.5.0;
2 contract Test {
3     function callsha256() public pure returns( bytes32 result){
4         return sha256("ronaldo");
5     }
6     function callkeccak256() public pure returns( bytes32 result){
7         return keccak256("ronaldo");
8     }
9 }
```

Low level interactions:

- CALldata:** A button labeled "Transact".
- TEST AT 0x332...D4B6D (MEMORY):**
 - callkeccak256:** Returns bytes32 result 0x2c96055cb975b6296 46e25b4c99b4530f0d9fa9123596b679b ceb0e5538eed13
 - callsha256:** Returns bytes32 result 0xe24dd2210803b4737a 9bd9e3163a4ca807b63201c3bc32b68fb 122a52eff96
- Low level interactions:** A button labeled "Transact".

Transaction Details:

- from: 0x58380a6a701c568545dcfc803fc8875f56bed44
- to: SolidityTest.(constructor)
- gas: 3000000 gas
- transaction cost: 114980 gas

At the bottom right, there's a search bar with placeholder text "Search with transaction hash or address".

Practical 7

Implement and demonstrate the use of the following in Solidity :

- (I).Contracts
- (II).Inheritance
- (III).Constructors
- (IV).Abstract Class
- (V).Interfaces

(I).Contracts

```
// Solidity program to  
// demonstrate how to  
// write a smart contract  
pragma solidity >= 0.4.16 < 0.7.0;
```

```
// Defining a contract
```

```
contract Test
```

```
{
```

```
// Declaring state variables
```

```
uint public var1;
```

```
uint public var2;
```

```
uint public sum;
```

```
// Defining public function
```

```
// that sets the value of
```

```
// the state variable

function set(uint x, uint y) public

{
    var1 = x;
    var2=y;
    sum=var1+var2;
}
```

```
// Defining function to
// print the sum of
// state variables

function get(
) public view returns (uint) {
    return sum;
}
```

Output:

SOLIDITY COMPILER

COMPILER: 0.5.17+commit.d19bba13

Include nightly builds

LANGUAGE: Solidity

EVM VERSION: compiler default

COMPILER CONFIGURATION

- Auto compile
- Enable optimization: 200
- Hide warnings

Compile smartcontract.sol

CONTRACT

Test (smartcontract.sol)

Publish on Swarm

Publish on Ipfs

Compilation Details

```

1 // Solidity program to
2 // demonstrate how to
3 // write a smart contract
4 pragma solidity >= 0.4.16 < 0.7.0;
5
6 // Defining a contract
7 contract Test {
8 +
9 // Declaring state variables
10 uint public var1;
11 uint public var2;
12 uint public sum;
13
14 // Defining public function
15 // that sets the value of
16 // the state variable
17 function set(uint x, uint y) public
18 {
19 +
20 var1 = x;
21 var2=y;
22 sum=var1+var2;
23 }
24
25 // Defining function to
26 // print the sum of
27 // state variables
28 function get(
29 + ) public view returns (uint) {
30 return sum;
31 }
32 }
33

```

from: 0x5b380a6a6701c560545dCfc803fc8075f56cedd4

to: SolidityTest.(constructor)

gas: 3000000 gas

DEPLOY & RUN TRANSACTIONS

0: bytes32 result 0xe24dd2210803b4737a9bd9e3163a4ca807b63201c3bc3b60fb122ca52eff36

Low level interactions

CALLDATA

Transact

TEST AT 0x5e1...4Eff5 (MEMORY)

set

x: 20
y: 15

get
sum
var1
var2

transact

```

1 // Solidity program to
2 // demonstrate how to
3 // write a smart contract
4 pragma solidity >= 0.4.16 < 0.7.0;
5
6 // Defining a contract
7 contract Test {
8 +
9 // Declaring state variables
10 uint public var1;
11 uint public var2;
12 uint public sum;
13
14 // Defining public function
15 // that sets the value of
16 // the state variable
17 function set(uint x, uint y) public
18 {
19 +
20 var1 = x;
21 var2=y;
22 sum=var1+var2;
23 }
24
25 // Defining function to
26 // print the sum of
27 // state variables
28 function get(
29 + ) public view returns (uint) {
30 return sum;
31 }
32 }
33

```

[vm] from: 0x5b3...edd4 to: Test.set(uint256,uint256) 0x5e1...4Eff5 value: 0 wei data: 0x1ab...0000f logs: 0 hash: 0x85f...143c6

status: true Transaction mined and execution succeed

transaction hash: 0x85fc5de9819161e66c32b59aab8e9c427b740d75c05a17394c2854b31a3c6

Debug

(II).Inheritance

```
// Solidity program to demonstrate Single Inheritance

pragma solidity >=0.4.22 <0.6.0;

// Defining contract
contract parent{

    // Declaring internal state variable
    uint internal sum;

    // Defining external function to set value of internal state variable sum
    function setValue() external {
        uint a = 10;
        uint b = 20;
        sum = a + b;
    }
}

// Defining child contract
contract child is parent{

    // Defining external function to return value of internal state variable sum
    function getValue() external view returns(uint) {
        return sum;
    }
}

// Defining calling contract
```

```
contract caller {  
  
    // Creating child contract object  
    child cc = new child();  
  
    // Defining function to call setValue and getValue functions  
    function testInheritance() public {  
        cc.setValue();  
    }  
  
    function result() public view returns(uint ){  
        return cc.getValue();  
    }  
}
```

Output:

The screenshot shows the Truffle UI interface for deploying and running transactions. On the left, there's a sidebar with icons for file operations, account management, and network selection. The main area is divided into several sections:

- DEPLOY & RUN TRANSACTIONS**: Includes fields for **ENVIRONMENT** (set to **JavaScript VM**), **ACCOUNT** (set to **0x5B3...eddC4 (99.9999999)**), **GAS LIMIT** (**3000000**), and **VALUE** (**0 wei**). A **Deploy** button is present.
- CONTRACT**: Shows the deployed contract **child - Hierarchical.sol**.
- Transactions recorded**: A list of deployed contracts:
 - TYPES AT 0XD91...39138 (MEMORY)**
 - TEST AT 0XD8B...33FA8 (MEMORY)**
 - TYPES AT 0XDA0...42B53 (MEMORY)**
 - TEST AT 0X9D7...B5E99 (MEMORY)**
 - LEDGERBALANCE AT 0XD2A...FD005 (MEMORY)**
 - LEDGERBALANCE AT 0X332...D4B6D (MEMORY)**
 - CHILD AT 0X406...2CFBC (MEMORY)**
- Low level interactions**: A section for interacting with the deployed contract, showing a **Transact** button.
- Hierarchical.sol**: The Solidity code for the contract, which includes:
 - // Solidity program to demonstrate Single Inheritance
 - pragma solidity >=0.4.22 <0.6.0;
 - // Defining contract
 - contract parent{}
 - // Declaring internal state variable
 - uint internal sum;
 - // Defining external function
 - // to set value of internal state variable sum
 - function setValue() external {
 - uint a = 10;
 - uint b = 20;
 - sum = a + b;
 - }
 - // Defining child contract
 - contract child is parent{}
 - // Defining external function
 - // to return value of internal state variable sum
 - function getValue() external view returns(uint) {
 - return sum;
 - }

(III).Constructors

```
// Solidity program to demonstrate  
// creating a constructor  
pragma solidity ^0.5.0;
```

```
// Creating a contract  
contract constructorExample {
```

```
// Declaring state variable  
string str;
```

```
// Creating a constructor  
// to set value of 'str'  
constructor() public {  
    str = "GeeksForGeeks";  
}
```

```
// Defining function to  
// return the value of 'str'  
function getValue()  
) public view returns (  
    string memory) {  
    return str;  
}
```

Output:

The screenshot shows the Solidity Compiler interface. On the left, there's a sidebar with various icons. The main area has tabs for "Arithmetic operator.sol", "Smart contract.sol", and "Contract constructor.sol".

SOLIDITY COMPILER

- COMPILER:** 0.5.17+commit.d19bba13
- Include nightly builds
- LANGUAGE:** Solidity
- EVM VERSION:** compiler default
- COMPILER CONFIGURATION:**
 - Auto compile
 - Enable optimization 200
 - Hide warnings

Compile Contract constructor.sol

CONTRACT: constructorExample (Contract constructorExample)

Actions:

- Publish on Swarm**
- Publish on Ipfs**
- Compilation Details**

ABI Bytecode

```
// Solidity program to demonstrate
// creating a constructor
pragma solidity ^0.5.0;

// Creating a contract
contract constructorExample {

    // Declaring state variable
    string str;

    // Creating a constructor
    // to set value of 'str'
    constructor() public {
        str = "GeeksForGeeks";
    }

    // Defining function to
    // return the value of 'str'
    function getValue()
    public view returns (
        string memory) {
        return str;
    }
}
```

DEPLOY & RUN TRANSACTIONS

ENVIRONMENT
JavaScript VM

ACCOUNT
0x5B3...eddC4 (99.9999999)

GAS LIMIT
3000000

VALUE
0 wei

CONTRACT
constructorExample - Contract constru

Deploy

Publish to IPFS

OR

At Address Load contract from Address

Transactions recorded (6)

Deployed Contracts

- SOLIDITYTEST AT 0xD91...39138 (MEMORY)
- SOLIDITYTEST AT 0xD8B...33FA8 (MEMORY)
- SOLIDITYTEST AT 0xF8E...9FBE8 (MEMORY)
- TEST AT 0xD7A...F771B (MEMORY)

CONSTRUCTOREXAMPLE AT 0xDA0...4

getValue

0: string: GeeksForGeeks

Low level interactions

CALldata

Transact

Arithmetic operator.sol

```
1 // Solidity program to demonstrate
2 // creating a constructor
3 pragma solidity ^0.5.0;
4
5 // Creating a contract
6 contract constructorExample {
7
8 // Declaring state variable
9 string str;
10
11 // Creating a constructor
12 // to set value of 'str'
13 constructor() public {
14 str = "GeeksForGeeks";
15 }
16
17 // Defining function to
18 // return the value of 'str'
19 function getValue(
20 ) public view returns (
21 string memory) {
22 return str;
23 }
```

Smart contract.sol

Contract constructor.sol

Practical 8

Implement and demonstrate the use of the following in Solidity :

(I).Libraries

(II).Assembly

(III).Events

(IV).Error Handling

(IV).Error Handling

```
// Solidity program to demonstrate require statement
```

```
pragma solidity ^0.5.0;
```

```
// Creating a contract
```

```
contract requireStatement {
```

```
// Defining function to check input
```

```
function checkInput(
```

```
uint _input) public view returns(
```

```
string memory){
```

```
require(_input >= 0, "invalid uint8");
```

```
require(_input <= 255, "invalid uint8");
```

```
return "Input is Uint8";
```

```
}
```

```
// Defining function to use require statement
```

```
function Odd(uint _input) public view returns(bool){
```

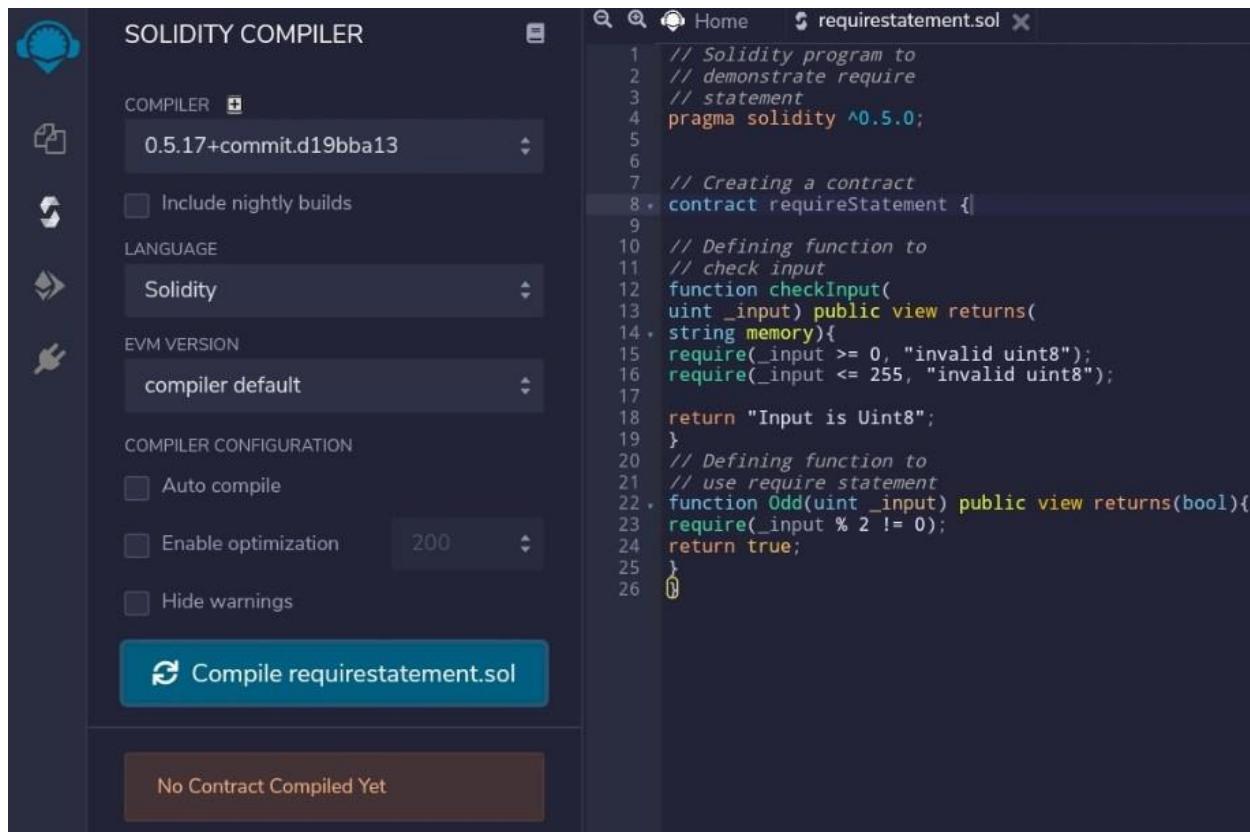
```
require(_input % 2 != 0);
```

```
return true;
```

```
}
```

}

Output:



The image shows the Solidity Compiler interface. On the left, there's a sidebar with icons for Home, Compiler, Languages, and EVM Version. The Compiler section is active, showing version 0.5.17+commit.d19bba13, a checkbox for 'Include nightly builds' (unchecked), and dropdown menus for Language (Solidity) and EVM Version (compiler default). Under Compiler Configuration, there are checkboxes for 'Auto compile' (unchecked), 'Enable optimization' (unchecked with a value of 200), and 'Hide warnings' (unchecked). A large blue button at the bottom says 'Compile requirestatement.sol'. Below it, a brown box displays the message 'No Contract Compiled Yet'. On the right, the file 'requirestatement.sol' is open in a code editor. The code is as follows:

```
// Solidity program to
// demonstrate require
// statement
pragma solidity ^0.5.0;

// Creating a contract
contract requireStatement {

    // Defining function to
    // check input
    function checkInput(
        uint _input) public view returns(
            string memory){
        require(_input >= 0, "invalid uint8");
        require(_input <= 255, "invalid uint8");

        return "Input is Uint8";
    }

    // Defining function to
    // use require statement
    function Odd(uint _input) public view returns(bool){
        require(_input % 2 != 0);
        return true;
    }
}
```

DEPLOY & RUN TRANSACTIONS

ENVIRONMENT: JavaScript VM

ACCOUNT: 0x5B3...eddC4 (99.9999999)

GAS LIMIT: 3000000

VALUE: 0 wei

CONTRACT: requireStatement - requirestatement.sol

Deploy

Publish to IPFS

OR

At Address Load contract from Address

Transactions recorded (1)

Deployed Contracts

REQUIRESTATEMENT AT 0xD91...3913

checkInput 254
0: string: Input is Uint8

Odd 253
0: bool: true

Low level interactions

CALLDATA

Transact

```

1 // Solidity program to
2 // demonstrate require
3 // statement
4 pragma solidity ^0.5.0;
5
6
7 // Creating a contract
8 contract requireStatement {
9
10 // Defining function to
11 // check input
12 function checkInput(
13     uint _input) public view returns(
14     string memory){
15     require(_input >= 0, "invalid uint8");
16     require(_input <= 255, "invalid uint8");
17
18     return "Input is Uint8";
19 }
20 // Defining function to
21 // use require statement
22 function Odd(uint _input) public view returns(bool){
23     require(_input % 2 != 0);
24     return true;
25 }
26

```

```
// Solidity program to demonstrate assert statement
```

```
pragma solidity ^0.5.0;
```

```
// Creating a contract
```

```
contract assertStatement {
```

```
// Defining a state variable
```

```
bool result;
```

```
// Defining a function to check condition
```

```
function checkOverflow(
```

```

    uint _num1, uint _num2) public {
        uint8 sum = _num1 + _num2;
        assert(sum<=255);
        result = true;
    }
}

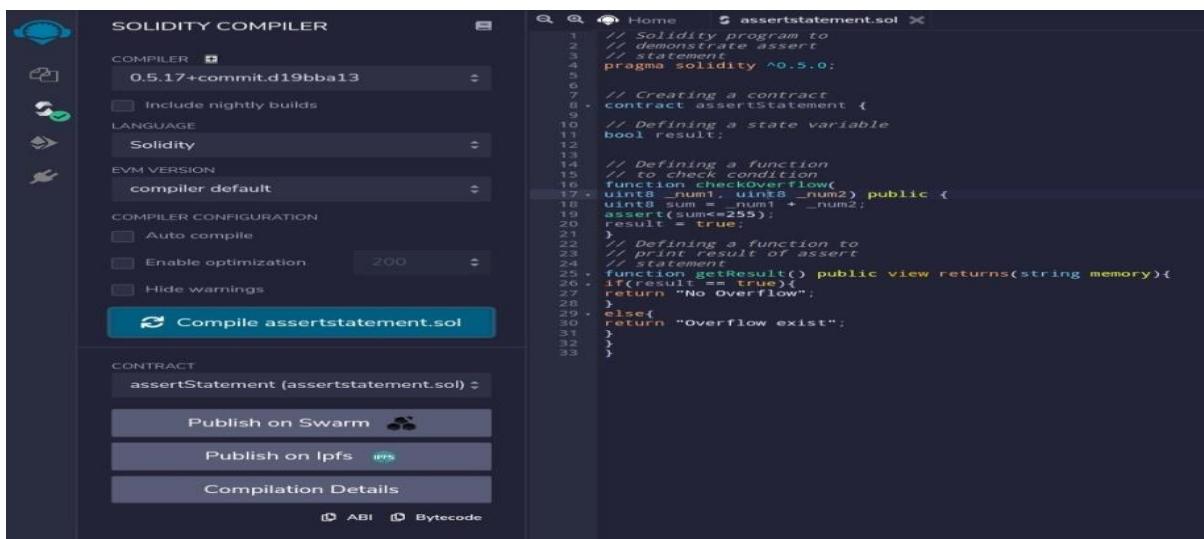
```

```

// Defining a function to print result of assert statement
function getResult() public view returns(string memory){
    if(result == true){
        return "No Overflow";
    }
    else{
        return "Overflow exist";
    }
}
}

```

Output:



The screenshot shows the Solidity Compiler interface. On the left, the 'SOLIDITY COMPILER' sidebar is visible with settings for the compiler version (0.5.17+commit.d19bba13), language (Solidity), EVM version (compiler default), and compiler configuration (Auto compile, Enable optimization set to 200). Below the sidebar, the 'CONTRACT' section displays the file 'assertStatement (assertstatement.sol)'. The code editor on the right contains the Solidity code provided above. The status bar at the bottom shows ABI and Bytecode options.

```

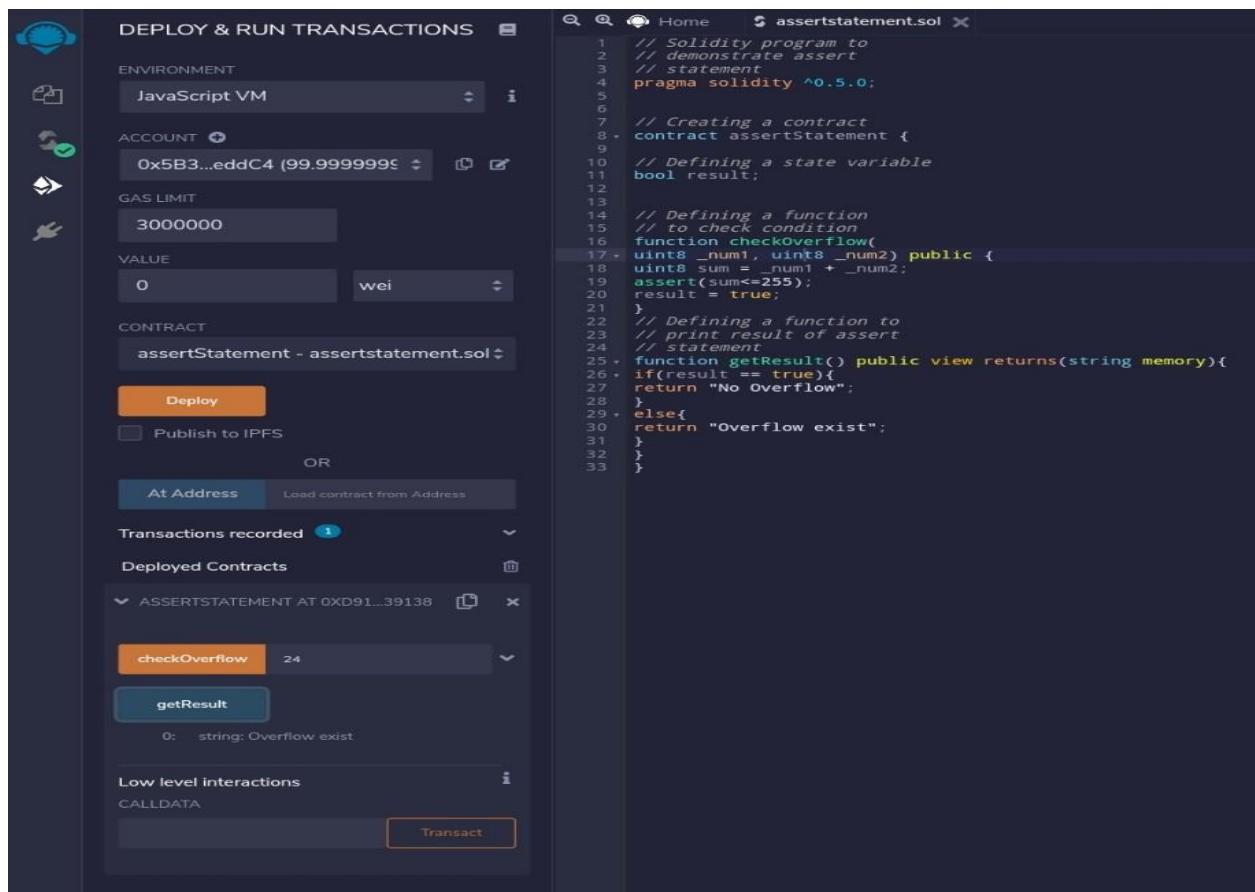
// Solidity program to demonstrate assert statement
pragma solidity ^0.5.0;

// Creating a contract
contract assertStatement {
    // Defining a state variable
    bool result;

    // Defining a function to check condition
    function checkOverflow(
        uint8 _num1, uint8 _num2) public {
        uint8 sum = _num1 + _num2;
        assert(sum<=255);
        result = true;
    }

    // Defining a function to print result of assert statement
    function getResult() public view returns(string memory){
        if(result == true){
            return "No Overflow";
        }
        else{
            return "Overflow exist";
        }
    }
}

```



// Solidity program to demonstrate assert statement

```
pragma solidity ^0.5.0;
```

// Creating a contract

```
contract assertStatement {
```

```
// Defining a state variable
```

```
bool result;
```

```
// Defining a function
```

```
// to check condition
```

```
function checkOverflow(uint8 sum) public {
```

```

assert(sum<=255);

result = true;

}

// Defining a function to print result of assert statement

function getResult() public view returns(string memory){

if(result == true){

return "No Overflow";

}

else{

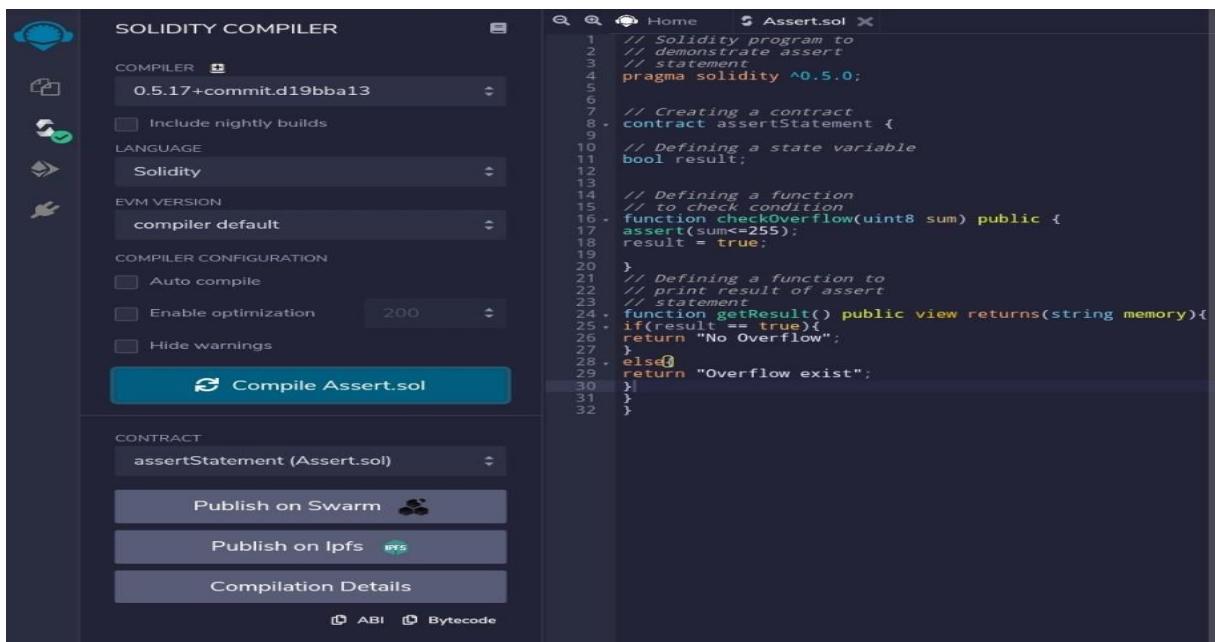
return "Overflow exist";

}

}

```

Output:



The screenshot shows the Solidity Compiler interface with the following configuration:

- COMPILER:** 0.5.17+commit.d19bba13
- LANGUAGE:** Solidity
- EVM VERSION:** compiler default
- COMPILER CONFIGURATION:**
 - Auto compile
 - Enable optimization: 200
 - Hide warnings
- CONTRACT:** assertStatement (Assert.sol)
- Buttons:** Publish on Swarm, Publish on Ipfs, Compilation Details, ABI, Bytecode

The code editor on the right displays the Solidity source code for `Assert.sol`:

```

1 // Solidity program to
2 // demonstrate assert
3 // statement
4 pragma solidity ^0.5.0;
5
6
7 // Creating a contract
8 contract assertStatement {
9 // Defining a state variable
10 bool result;
11
12
13
14 // Defining a function
15 // to check condition
16 function checkOverflow(uint8 sum) public {
17 assert(sum<=255);
18 result = true;
19 }
20
21 // Defining a function to
22 // print result of assert
23 // statement
24 function getResult() public view returns(string memory){
25 if(result == true){
26 return "No Overflow";
27 }
28 else{
29 return "Overflow exist";
30 }
31 }
32 }

```

The screenshot shows the Truffle UI interface. On the left, the 'DEPLOY & RUN TRANSACTIONS' sidebar includes fields for ENVIRONMENT (JavaScript VM), ACCOUNT (0x5B3...eddC4), GAS LIMIT (3000000), and VALUE (0 wei). Below these are sections for CONTRACT (assertStatement - Assert.sol) and interaction buttons like Deploy, Publish to IPFS, and At Address. The right side displays the Solidity code for 'Assert.sol' and its deployed state. The code defines a contract with an assert statement and two functions: checkOverflow and getResult. The getResult function returns "Overflow exist" if the result of the assert is false.

```

// Solidity program to demonstrate assert statement
pragma solidity ^0.5.0;

// Creating a contract
contract assertStatement {
    // Defining a state variable
    bool result;

    // Defining a function to check condition
    function checkOverflow(uint8 sum) public {
        assert(sum<=255);
        result = true;
    }

    // Defining a function to print result of assert
    function getResult() public view returns(string memory){
        if(result == true){
            return "No Overflow";
        } else{
            return "Overflow exist";
        }
    }
}

```

// Solidity program to demonstrate revert statement

```
pragma solidity ^0.5.0;
```

// Creating a contract

```
contract revertStatement {
```

// Defining a function to check condition

```
function checkOverflow(
```

```
uint _num1, uint _num2) public view returns(
```

```
string memory, uint) {
```

```
uint sum = _num1 + _num2;
```

```

if(sum < 0 || sum > 255){

revert(" Overflow Exist");

}

else{

return ("No Overflow", sum);

}

}

```

Output:

The screenshot shows the Solidity Compiler interface with the following details:

- Compiler:** Version 0.5.17+commit.d19bba13
- Language:** Solidity
- EVM Version:** compiler default
- Compiler Configuration:**
 - Auto compile
 - Enable optimization: 200
 - Hide warnings
- Contract:** revertStatement (revertstatement.sol)
- Buttons:** Publish on Swarm, Publish on Ipfs, Compilation Details
- Status Bar:** Shows a green checkmark icon, transaction details: from: 0x5B3...eddC4 to: Test.set(uint256,uint256) 0x5e1...4Efd5 value: 0 wei data: 0x1ab...0000f logs: 0 hash: 0x85f...143c6, and a Debug button.

The screenshot shows the Truffle UI interface for deploying and running transactions. On the left, there's a sidebar with icons for deployment, mining, and monitoring. The main area has tabs for different Solidity files: overloading.sol, cryptodata.sol, smartcontract.sol, and revertstatement.sol. The revertstatement.sol tab is active.

The code for revertstatement.sol is displayed:

```
// Solidity program to demonstrate revert statement
pragma solidity ^0.5.0;

// Creating a contract
contract revertStatement {
    // Defining a function to check condition
    function checkOverflow(
        uint _num1, uint _num2)
        public view returns(
            string memory, uint)
    {
        uint sum = _num1 + _num2;
        if(sum < 0 || sum > 255){
            revert("Overflow Exist");
        } else{
            return ("No Overflow", sum);
        }
    }
}
```

The UI shows a transaction being executed for the checkOverflow function with parameters _num1: 52 and _num2: 35. The result is "0: string: No Overflow".

At the bottom, the transaction status is shown as "true Transaction mined and execution succeed" with the transaction hash: 0x05fc...143c6. There is also a "Debug" button.